МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
вычислительной техники	МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

Веб-инструмент для создания и редактирования интерактивных диалогов в игровых приложениях

Студент _____ Антонов Андрей Владимирович
(подпись, дата) (ФИО полностью)

Руководитель _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____№_____)

Заведующий кафедрой _____ Козлов А.Ю.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____№____)

Секретарь ГЭК _____ Попова Н.А.
(подпись) (фамилия, инициалы)

Пенза, 2025

Федеральное государственное бюджетное образовательное учреждение высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
Вычислительной техники	МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____А.Ю. Козлов «20»
декабря 2024 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент Антонов Андрей Владимирович гр. 21ВП1 факультета ВТ
<Фамилия, имя, отчество полностью>
2.Тема работы Веб-инструмент для создания и редактирования интерактивных диалогов в игровых приложениях

Тема утверждена приказом ПГУ №_____от "___"_____20_____

3. Руководитель работы доцент кафедры «МОиПЭВМ», доцент, к.т.н., Такташкин Д.В.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

1) Требуется разработать веб-инструмент для создания и редактирования интерактивных диалогов в игровых приложениях.
2) Среда выполнения: операционная система Windows 10, Web-браузер
3) Языковые средства разработки: JavaScript, UML
4) Инструментальные средства: VS Code, Microsoft Project 2013, Visual Paradigm 17.2, Draw.io, NPM
5) Основные функции:
а) создание и редактирование интерактивных диалогов;
б) проверка структурной целостности диалогов;
в) экспортирование в JSON
г) импортирование JSON в вид диалогового графа;
д) управление проектом диалогов;
е) симуляция диалогов;
ж) экспортирование в формат изображения;

5. Перечень подлежащих разработке вопросов

1) Анализ предметной области и постановка задачи
2) Проектирование
3) Реализация
4) Контроль качества программного обеспечения
5) План разработки и оценка бюджета
6) Оформление пояснительной записки

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2025 – 14.04.2025	
2	Проектирование программных средств	20%	15.04.2025 – 25.04.2025	
3	Реализация программных средств	30%	26.04.2025 – 12.05.2025	
4	Тестирование программных средств	20%	13.05.2025 – 22.05.2025	
5	Оформление пояснительной записки	20%	23.05.2025 – 03.06.2025	

Дата выдачи задания «20» декабря 2024г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 117 листов, 35 рисунков, 9 таблиц, 20 использованных источников и 3 приложения.

ВЕБ-ПРИЛОЖЕНИЕ, ГРАФОВЫЙ РЕАДАКТОР, ДИАЛОГОВЫЕ СЦЕНАРИИ, JSON, UML, JAVASCRIPT, REACTJS, REACTFLOW

Объектом исследования являются веб-технологии и средства упрощения разработки диалоговых сценариев для игровых приложений.

Предметом исследования являются программные средства для создания, визуализации и управления интерактивными диалогами в виде графа с возможностью преобразования в удобный формат хранения данных.

Разработка приложения проводилась на языке программирования JavaScript с использованием библиотеки React и специализированной библиотеки React Flow. В качестве среды разработки использовалась Visual Studio Code.

Среда исполнения – Microsoft Windows 10 и выше, web-браузер.

Результаты работы – в ходе выполнения выпускной квалификационной работы был создан веб-инструмент интерактивного редактирования диалоговых систем в виде графа для игровых приложений.

Содержание

Введение 7

1 Постановка задачи и анализ требований веб-инструмента для создания и редактирования интерактивных диалогов в игровых приложениях 9

1.1 Анализ предметной области и постановка задачи 9

1.2 Обзор аналогов 10

1.2.1 Twine 10

1.2.2 Articy:draft 11

1.2.3 Draw.io 12

1.2.4 ChatMapper 13

1.2.5 YarnEditor 14

1.2.6 Сравнение продуктов-аналогов 15

1.3 Анализ функциональных требований 16

1.4 Анализ нефункциональных требований 20

2 Проектирование веб-инструмента по созданию и редактированию интерактивных диалогов в игровых приложениях 21

2.1 Архитектура приложения 21

2.2 Проектирование структуры хранения 24

2.3 Проектирование программного средства 26

2.4 Проектирование пользовательского интерфейса 30

2.5 Планирование разработки и оценка бюджета 33

3 Реализация программных средств веб-инструмента для создания и редактирования интерактивных диалогов в игровых приложениях 36

3.1 Реализация слоя данных приложения 36

3.2 Реализация слоя бизнес-логики приложения 39
3.3 Разработка слоя представления приложения 42
4 Тестирование модуля создания узлов 48
4.1 Тестирование создания узлов методом чёрного ящика 48
4.1.1 Классы эквивалентности 48
4.1.2 Функциональная диаграмма и таблица решений 49
4.2 Анализ сложности кода модуля создания узлов 54
4.3 Модульное тестирование 58
Заключение 61
Список использованных источников 63
Приложение А (справочное) Глоссарий 65
Приложение Б (обязательное) Листинг программы 68
Приложение В (рекомендуемое) Листинг автоматизированных тестов 111

Введение

В современном мире интерактивные системы, такие как чат-боты, диалоговые тренажеры, квесты, компьютерные игры и обучающие программы с разветвленными сценариями, становятся все более востребованными. Сложность проектирования и поддержки таких систем постоянно растет, предъявляя высокие требования к инструментам разработки. Визуализация структуры диалога или процесса является ключевым аспектом для понимания его логики, выявления узких мест и эффективного внесения изменений.

Традиционные подходы к проектированию сложных интерактивных сценариев, часто основанные на текстовых описаниях или блок-схемах в универсальных графических редакторах, не всегда обеспечивают необходимый уровень наглядности, специализации и удобства для быстрой итеративной разработки. Современные веб-технологии позволяют создавать специализированные программные средства, которые могут существенно упростить и ускорить процесс создания, редактирования и управления такими интерактивными системами.

Объект исследования – веб-технологии и средства упрощения разработки диалоговых сценариев для игровых приложений.

Предмет исследования – программные средства для создания, визуализации и управления интерактивными диалогами в виде графа с возможностью преобразования в удобный формат хранения данных.

Целью выпускной квалификационной работы является разработка веб-инструмента для визуального конструирования и управления графами диалоговых сценариев, обеспечивающих наглядное представление структуры и удобные инструменты для их модификации.

Для достижения поставленной цели необходимо выполнить следующие задачи:

– провести сравнительный анализ продуктов-аналогов;

– выявить и проанализировать требования к разрабатываемому приложению;

- спроектировать программные средства и структуры данных;
- провести планирование и расчет бюджета проекта;
- осуществить разработку программного продукта;
- провести тестирования разработанного приложения;
- оформить пояснительную записку в соответствии с требованиями.

Веб-инструмент для создания и редактирования интерактивных диалогов в игровых приложениях повысит эффективность процесса проектирования и управления интерактивными диалоговыми системами, обеспечивая разработчикам и сценаристам удобный инструмент для создания сложных и разветвленных диалогов.

1 Постановка задачи и анализ требований веб-инструмента для создания и редактирования интерактивных диалогов в игровых приложениях

1.1 Анализ предметной области и постановка задачи

Интерактивные диалоговые системы стали неотъемлемой частью развлекательного сегмента интернета. Эти системы базируются на заранее спроектированных сценариях взаимодействия, которые могут достигать высокой степени сложности и нелинейности. Качество и эффективность таких систем напрямую зависят от продуманности и логической сложности заложенных в них диалоговых веток.

Проектирование таких сценариев часто представляет собой сложную задачу. Диалоговый сценарий, по своей сути, является графом, который должен состоять из трёх главных структурных элементов:

– узлы, представляющие собой определенные состояния или шаги в диалоге;

– рёбра, являющиеся переходами между состояниями, обусловленные выбором пользователя;

– реакции внутри узлов, которые определяют возможные пути развития диалога.

Классические подходы проектирования могут быть трудоемкими, недостаточно наглядными для сложных структур и плохо подходить для командной разработки. Возникает потребность в специализированном инструменте, который бы позволил визуально конструировать, редактировать и анализировать такие диалоговые графы.

Веб-инструмент для создания и редактирования интерактивных диалогов в игровых приложениях должна представлять собой специализированную систему, которая обеспечивает интуитивно понятное управление процессом проектирования диалоговых сценариев. Цель данного инструмента – создание эффективной среды, позволяющей сократить время на разработку диалоговых

систем.

Программное средство должно иметь понятный пользовательский интерфейс, корректно отображающий текущую структуру диалога, не вызывая задержки при рендеринге графа.

1.2 Обзор аналогов

Для определения текущего состояния рынка и выявления сильных и слабых сторон существующих решений был проведен анализ программных средств, обладающих схожим или пересекающимся функционалом с разрабатываемым веб-инструментом для создания и редактирования интерактивных диалогов.

1.2.1 Twine

Twine – это бесплатный инструмент с открытым исходным кодом для создания интерактивных, нелинейных историй и текстовых игр. Он позволяет визуально связывать текстовые блоки для формирования разветвленных повествований. Поддерживает использование переменных, условных переходов и макросов для добавления сложной логики [1]. Интерфейс приложения приведен на рисунке 1.

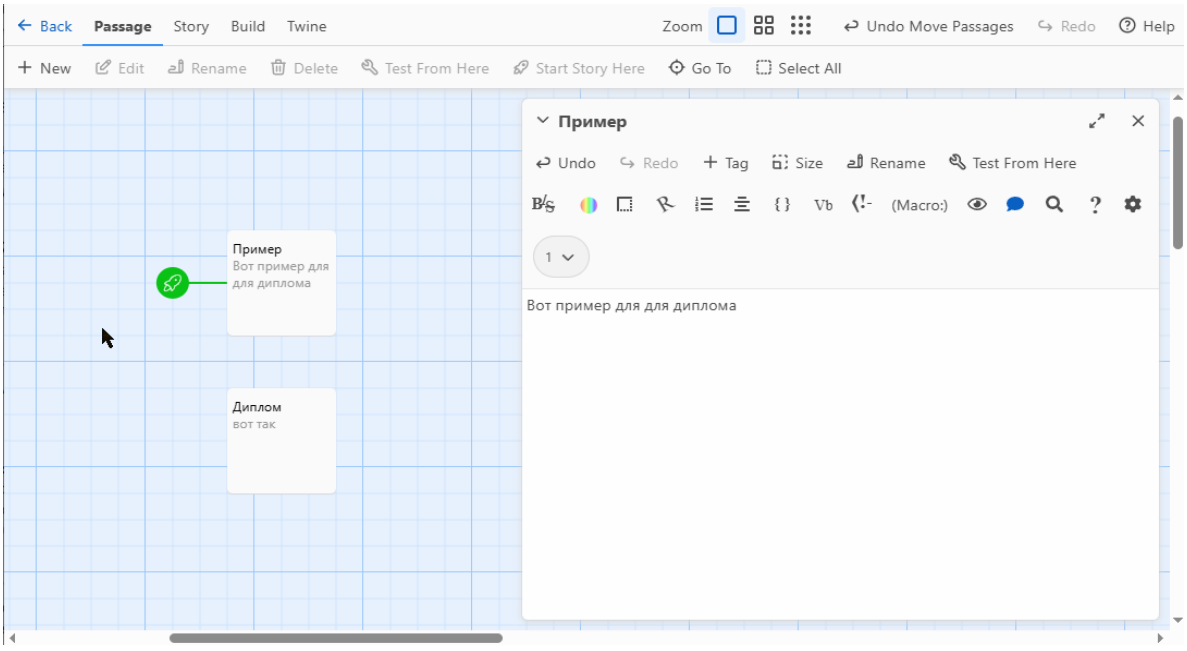

Рисунок 1 – Интерфейс рабочего пространства «Twine»

К достоинствам можно отнести возможно не писать код, чтобы создать простую историю, но можно расширить свои истории переменными, условной логикой, изображениями, CSS и JavaScript.

К недостаткам можно отнести то, что нет удобной площадки для распространения. В случае с большими историями дерево игры получается очень громоздким. Таким образом, сложности с дистрибуцией контента и визуализацией объемных проектов являются ключевыми ограничениями данного инструмента.

1.2.2 Articy:draft

Articy:draft – это коммерческий инструмент, предназначенный для сценаристов, геймдизайнеров и нарративных дизайнеров. Он позволяет создавать сложные сюжетные структуры, диалоги, базы данных персонажей, локаций и предметов, а также визуализировать потоки игрового процесса. Предусматривает экспорт данных в различные форматы для интеграции с игровыми движками [2]. Интерфейс приложения приведен на рисунке 2.

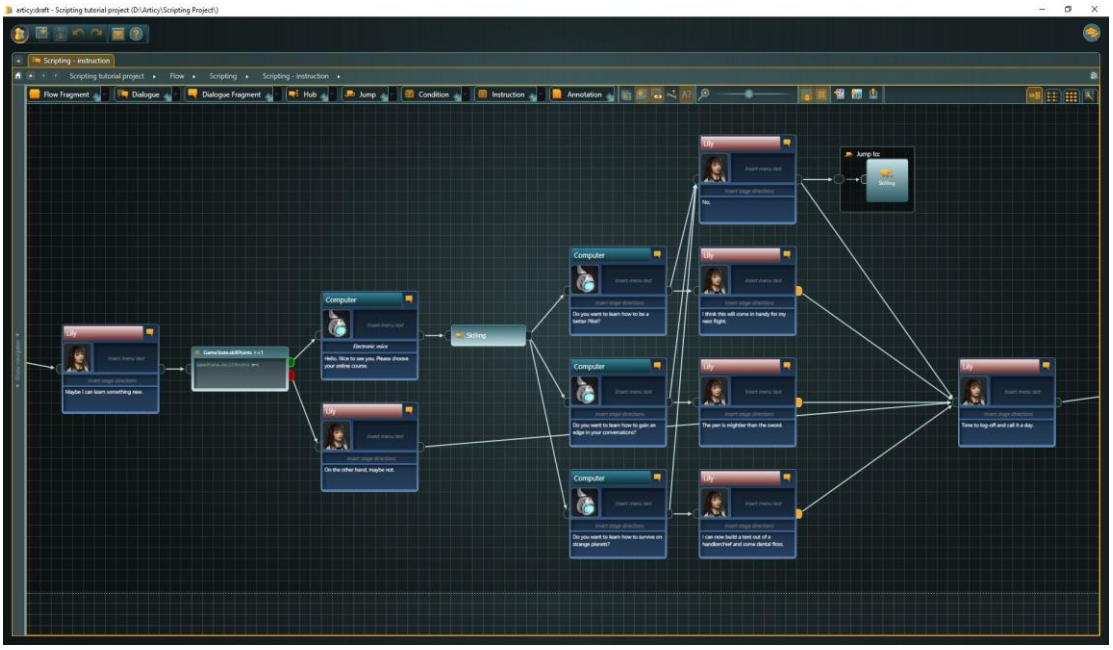

Рисунок 2 – Интерфейс рабочего пространства «Articy:draft»

К достоинствам можно отнести глубокие возможности по настройке объектов и их связей, инструменты для командной работы предоставление

интеграции с популярными игровыми движками.

К недостаткам можно отнести отсутствие системы защиты от потерь данных во время сбоев системы. Следовательно, отсутствие встроенных механизмов защиты от потери данных при системных сбоях является значимым упущением.

1.2.3 Draw.io

Draw.io – это бесплатный онлайн-инструмент для создания различных типов диаграмм, включая блок-схемы, UML-диаграммы, организационные диаграммы и т. д. Обладает широким набором стандартных фигур и коннекторов, позволяет настраивать их внешний вид и сохранять диаграммы в различных форматах, включая интеграцию с облачными хранилищами [3]. Интерфейс приложения приведен на рисунке 3.

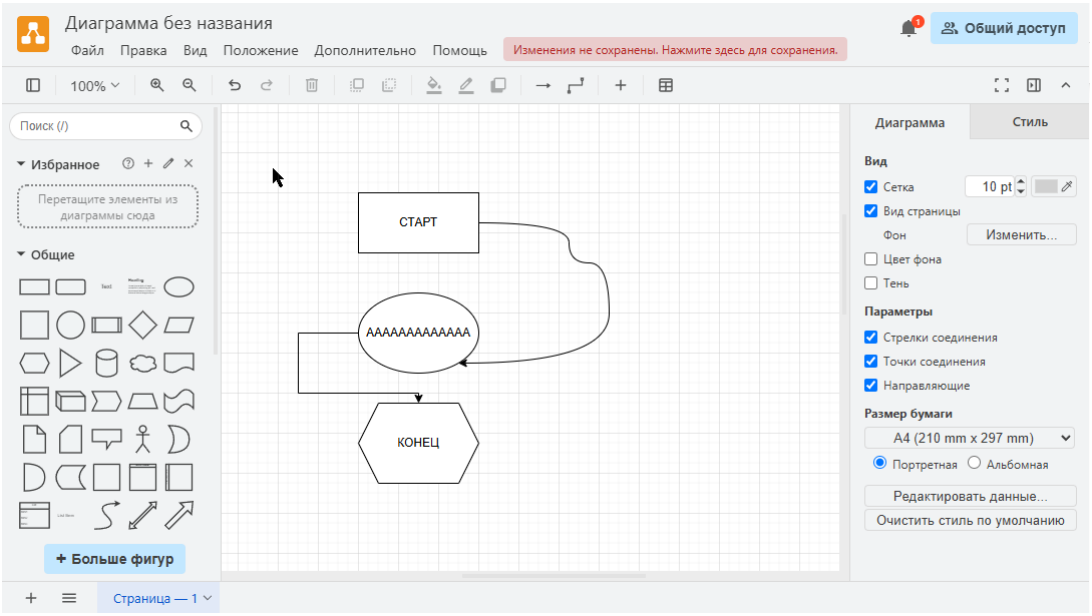

Рисунок 3 – Интерфейс рабочего пространства «Draw.io»

К достоинствам можно отнести бесплатную доступность полного функционала и доступность через веб-браузер. Кроме того, платформа имеет огромное количество готовых пресетов и гибкую систему настройки всех элементов.

Главным недостатком является то, что draw.io не специализирован на построении интерактивных диалогов, что принесёт трудности обработки

экспортируемых данных для интеграции в игровые приложения. Таким образом, данный продукт не специализируется на создании интерактивных диалогов и, как следствие, потенциальные сложности с интеграцией экспортируемых данных в игровые проекты, являются его основным ограничением в данном контексте.

1.2.4 ChatMapper

ChatMapper – это коммерческий инструмент, специально разработанный для создания и тестирования нелинейных диалогов для игр и других интерактивных приложений. Позволяет визуально проектировать диалоговые деревья, использовать переменные, условия, скрипты на Lua, а также эмулировать диалоги для проверки их логики [4]. Интерфейс приложения приведен на рисунке 4.

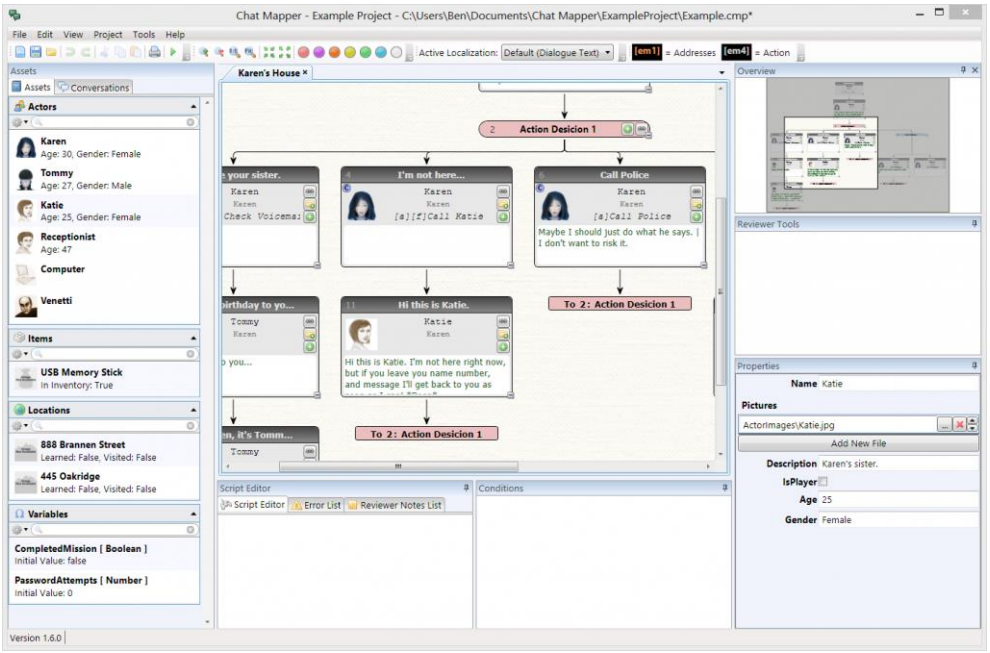

Рисунок 4 – Интерфейс рабочего пространства «ChatMapper»

К главному достоинству можно отнести встроенный симулятор диалогов и возможность экспорта в различные форматы, включая XML и JSON.

Недостатками являются устаревший интерфейс и платный функционал. Таким образом, устаревший пользовательский интерфейс в сочетании с коммерческой моделью распространения ограничивают его доступность и

удобство использования для широкого круга разработчиков.

1.2.5 YarnEditor

Yarn Editor – это редактор диалогов с открытым исходным кодом, созданный для игры «Night in the Woods» и впоследствии развиваемый сообществом. Он предназначен для написания разветвленных диалогов и нарративов для игр. Инструмент использует простой текстовый синтаксис, похожий на Twine, для определения узлов диалога, команд и условных переходов. Редактор доступен как веб-приложение, а также в виде десктопных сборок [5]. Интерфейс приложения приведен на рисунке 5.

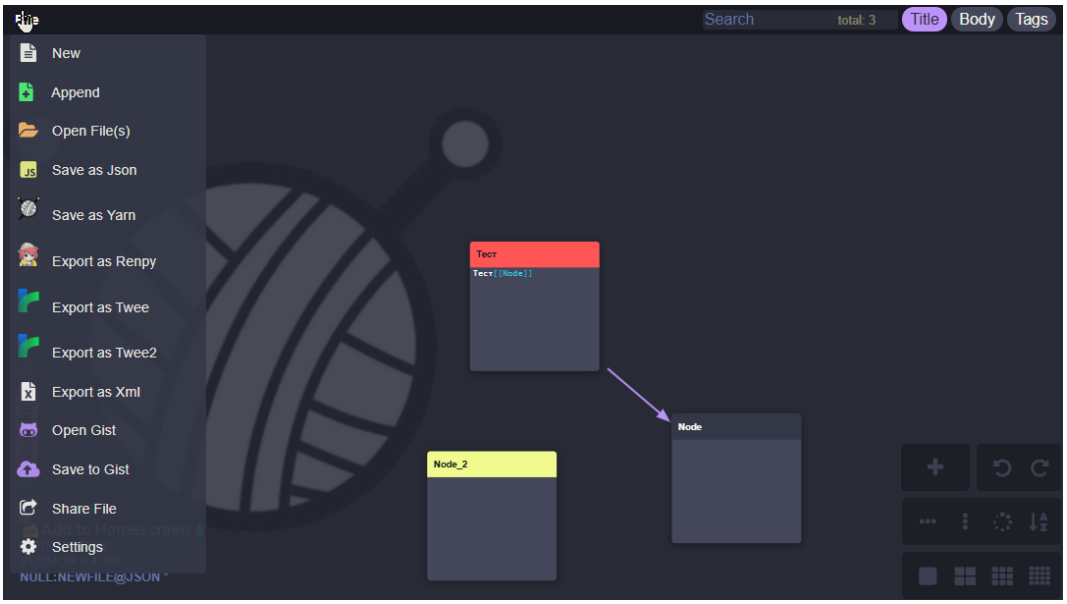

Рисунок 5 – Интерфейс рабочего пространства «YarnEditor»

К достоинствам можно отнести поддержку различных форматов экспорта, открытый исходный код и ориентированность на создание диалогов для игр с простой логикой.

К недостаткам можно отнести то, что способ создания связей и логики является текстовым. Следовательно, текстовый способ определения связей и логики может снижать наглядность и интуитивность процесса для пользователей, предпочитающих визуальное конструирование.

1.2.6 Сравнение продуктов-аналогов

Проведем сравнительный анализ рассмотренных выше программных средств.

Для оценки будет использоваться трехбалльная система оценки, в которой 0 означает «Неудовлетворительно», 1 – «Удовлетворено частично», а 2 – «Удовлетворительно» рассматриваемой функции. Результаты сравнения приведены в таблице 1.

Таблица 1 – Сравнение программных средств аналогичного назначения

Критерий оценивания	Yarn Editor	Twine	Articy:draft	Draw.io	ChatMapper
Специализация на интерактивных диалогах	2	1	2	0	2
Визуальное конструирование графа	1	2	2	2	2
Гибкая настройка узлов	2	1	2	2	2
Возможность экспорта в легко интерпретируемый формат данных	2	1	2	0	2
Доступность	1	2	0	2	0
Возможность внедрения скриптов	1	1	2	0	2
Интуитивно понятный интерфейс	1	0	1	2	0
ИТОГО	3	4	7	3	5

Таким образом можно сделать вывод, что ни одно из рассмотренных приложений не соответствует объявленным критериям в полной мере.

Коммерческие продукты, такие как Articy:draft и ChatMapper, предлагают богатый функционал, однако их стоимость и сложность могут оттолкнуть пользователя. Бесплатные и открытые инструменты, такие как Twine и Yarn Spinner Editor, предоставляют хороший набор возможностей для создания игровых диалогов, но могут иметь свои особенности в подходе к визуальному конструированию. Универсальные инструменты, вроде Draw.io, требуют значительной адаптации.

В результате анализа принято решение о целесообразности разработки собственного веб-инструмента. Разрабатываемое приложение будет стремиться объединить интуитивность и доступность редактора графов, ориентированного на интерактивное взаимодействие, что является ключевым отличием от текстового подхода Yarn Editor. При этом сохраняется фокус на легковесности, удобстве использования и возможности экспорта данных в распространенные форматы, предоставляя разработчикам игр простой, но эффективный инструмент.

1.3 Анализ функциональных требований

На основе анализа предметной области и поставленных задач были сформулированы следующие функциональные требования к разрабатываемому веб-инструменту для создания и редактирования интерактивных диалогов для игровых приложений:

- создать проект;
- открыть существующий проект;
- создать новый диалог;
- открыть существующий диалог;
- создать логику взаимодействия между диалогами;
- открыть диалог для редактирования;
- добавить на холст узел определенного типа;
- создать связь между узлами;
- добавить реакцию у высказывания;

– редактировать атрибуты узла;
– редактировать атрибуты реакции у высказывания;
– удалить узел;
– удалить реакцию;
– сохранить текущий проект;
– загрузить сохраненный проект;
– экспортировать текущий проект в выбранный формат;
– импортировать проект из файла поддерживаемого формата;
– осуществлять навигацию по холсту.

В соответствии с поставленной задачей и выявленными функциональными требованиями была разработана диаграмма вариантов использования [6], которая представлена на рисунке 6.

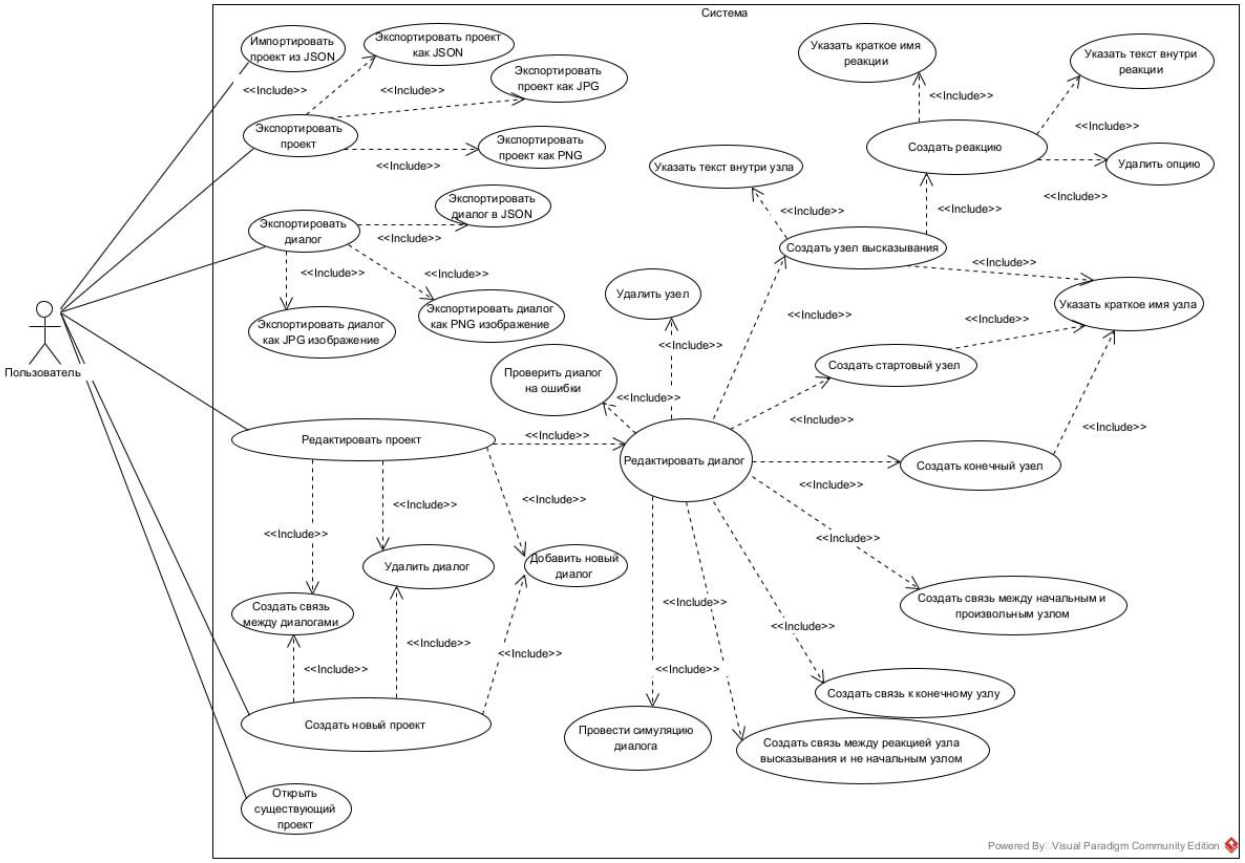

Рисунок 6 – Диаграмма вариантов использования инструмента создания диалогов

Спецификация прецедента «Проверить диалог на ошибки» представлена на рисунке 7.

Прецедент: Проверить диалог на ошибки
ID: 1
Краткое описание: Система указывает пользователю на ошибки в диалоге
Главные актеры: Пользователь
Предусловие: Диалог создан
Основной поток: 1. Пользователь нажимает кнопку для проверки диалога 2. Система открывает отдельное окно 3. Система выводит список ошибок на экран 4. Система подсвечивает элементы, содержащие ошибки, если такие есть. Альтернативные потоки: 2.а. Система выводит сообщение что ошибок не обнаружено
Постусловия: Система указала все ошибки

Рисунок 7 – Спецификация прецедента «Проверить диалог на ошибки»

Спецификация прецедента «Открыть существующий проект» представлена на рисунке 8.

Прецедент: Открыть существующий проект
ID: 2
Краткое описание: Пользователь открывает сохранённый в системе проект
Главные актеры: Пользователь
Предусловие: Пользователь был авторизован
Основной поток: 1. Пользователь открывает «проект» 2. Пользователь нажимает кнопку «открыть» 3. Пользователь выбирает в открывшемся файловом менеджере нужный ему файл
Постусловия: Интерактивный диалог из открытого проекта выведен на экран.

Рисунок 8 – Спецификация прецедента «Открыть существующий проект»

Спецификация прецедента «Создать связь между реакцией узла высказывания и не начальным узлом» представлена на рисунке 9.

Прецедент: Создать связь между реакцией узла высказывания и целевым узлом
ID: 3
Краткое описание: Пользователь добавляет переход от реакции узла высказывания и любым узлом кроме начального.
Главные актеры: Пользователь
Предусловие: В созданном проекте существует узел высказывания с хотя-бы одной реакций и целевой узел, который не может быть начальным
Основной поток: 1. Пользователь нажимает на узел 2. Пользователь, зажав левую кнопку мыши, перетаскивает графическую стрелку к целевому узлу 3. Пользователь отпускает курсор мыши над целевым узлом
Постусловия: Связь успешно создана и графически отображена на холсте

Рисунок 9 – Спецификация прецедента «Создать связь между реакцией узла высказывания и целевым узлом»

Спецификация прецедента «Создать узел высказывания» представлена на рисунке 10.

Прецедент: Создать узел высказывания
ID: 4
Краткое описание: Пользователь размещает на холсте узел высказывания
Главные актеры: Пользователь
Предусловие: Проект создан
Основной поток: 1. Пользователь нажимает на узел высказывания в левой боковой панели элементов 2. Пользователь, зажав левую кнопку мыши, перетаскивает иконку узла высказывания на холст 3. Пользователь отпускает курсор мыши на холсте
Постусловия: Узел высказывания создан и отображен на холсте

Рисунок 10 – Спецификация прецедента «Создать узел высказывания»

В результате анализа функциональных требований были выделены варианты использования программы. Для основных вариантов использования

были представлены их спецификации, которые позволяют отобразить последовательность действий пользователя и системы при выполнении определенных функций.

1.4 Анализ нефункциональных требований

Анализ нефункциональных требований для разрабатываемого приложения включает в себя рассмотрение нескольких аспектов.

Приложение обязано обеспечивать отзывчивое взаимодействие пользователя при работе с диалоговыми графами. При создании и перемещении узлов, связей и масштабировании не должно возникать заметных задержек.

Интерфейс должен быть интуитивно понятным для пользователя. Пользователи должны легко осваивать основные функции: добавление узлов разных типов, их связывание, редактирование свойств узлов и реакций. Процесс импорта и экспорта должен быть простым и очевидным.

Приложению необходимо стабильно работать, сохранять данные пользователя без потерь и корректно обрабатывать ошибочные ситуации. Данные графа должны корректно сохраняться при закрытии и восстанавливаться при следующем открытии веб-инструмента.

Веб-инструмент должен корректно функционировать в последних стабильных версиях следующих веб браузеров: Google Chrome, Mozilla Firefox, Safari, Microsoft Edge. Экспортируемые файлы проекта должны иметь четко определенную структуру, пригодную для чтения и интеграции с игровыми приложениями.

Все эти аспекты будут учтены при разработке и реализации проекта для обеспечения его успешной работы и удовлетворения потребностей пользователей.

2 Проектирование веб-инструмента по созданию и редактированию интерактивных диалогов в игровых приложениях

2.1 Архитектура приложения

Ключевым архитектурным решением при проектировании веб-инструмента для создания и редактирования интерактивных диалогов является его реализация в виде одностраничного приложения (Single Page Application, SPA). Такой подход обеспечивает быструю и плавную работу интерфейса, так как большинство ресурсов (HTML, CSS, JavaScript) загружаются один раз, а последующие переходы между разделами и обновления данных происходят динамически без полной перезагрузки страницы [7]. Для управления представлениями и навигацией в рамках SPA будет использована библиотека react-router-dom, позволяющая организовать логическую структуру различных экранов и переходов между ними [8].

В рамках архитектуры SPA, приложение будет логически структурировано с применением принципов многослойности для обеспечения разделения ответственности и повышения модульности. Выделяются следующие основные слои: слой представления, слой управления состоянием и бизнес-логики, и слой данных.

Слой представления отвечает за взаимодействие с пользователем и отображение информации. Его реализация будет основана на компонентном подходе с использованием библиотеки React [9] и React Flow [10]. Этот слой будет включать компоненты для визуализации и редактирования графовых структур диалогов, навигационные элементы, управляемые с помощью react-router-dom, и формы для ввода и отображения данных. Основной задачей этого слоя является предоставление интуитивно понятного и отзывчивого пользовательского интерфейса, который отвечает поставленным функциональным и не функциональным требованиям.

Слой бизнес-логики инкапсулирует основные правила и операции приложения. В текущей концепции, применимой для SPA с обработкой всего

кода программного приложения на одной странице, этот слой будет интегрирован в управляющие компоненты React. Он будет отвечать за управление состоянием приложения, обработку пользовательских действий, валидацию данных, а также за координацию взаимодействия между слоем представления и слоем данных.

Слой данных обеспечивает хранение и доступ к информации. Этот слой будет предоставлять интерфейс для выполнения операций сохранения, загрузки и модификации данных, таких как проекты, совокупности диалогов и их компонентов, различные настройки узлов.

Для визуализации компонентов, входящих в состав приложения, можно использовать диаграмму компонентов, построенную с помощью UML-средств. Она позволяет более четко представить структуру веб-инструмента и его архитектуру. Диаграмма компонентов позволяет точно определить архитектуру разрабатываемой системы, установив понятные зависимости между программными компонентами, в роли которых может выступать исполняемый код. Во многих средах разработки модуль или компонент соответствует файлу. Пунктирные стрелки, соединяющие модули, показывают отношения взаимозависимости, аналогичные тем, которые имеют место при компиляции исходных текстов программ. Основными графическими элементами диаграммы компонентов являются компоненты, интерфейсы и зависимости между ними. На текущей диграмме представлены основные три слоя разрабатываемого веб-приложения, в каждом слое представлены эксизные варианты проектируемых файлов, название которых соответствует их выполняемой функции и слою архитектуры.

Диаграмма компонентов обеспечивает согласованный переход от логического представления к конкретной реализации проекта в форме программного кода. Одни компоненты могут существовать только на этапе компиляции программного кода, другие – на этапе его исполнения [11].

На рисунке 11 изображена диаграмма компонентов приложения.

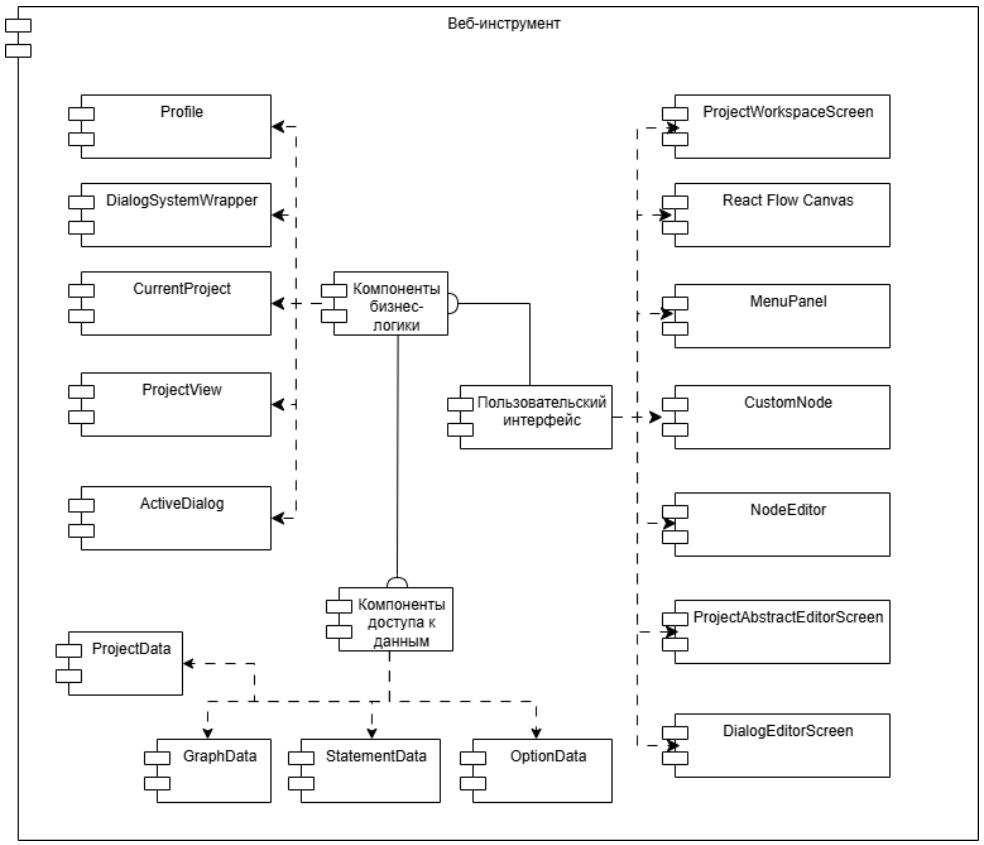

Рисунок 11 – Диаграмма компонентов веб-инструмента

Итогом этапа проектирования стала выработка трехуровневой архитектурной модели.

Обработка запросов или выполнение операций в приложении зачастую подразумевает последовательное прохождение через все эти слои. Такая архитектурная организация обладает рядом преимуществ:

– относительная простота в понимании и реализации;

– высокий уровень абстракции, позволяющий изменять реализацию одного слоя без существенного влияния на другие;

– повышенная безопасность за счет изоляции отдельных слоев друг от друга;

– улучшенная управляемость и потенциал для масштабирования за счет четкого разделения компонентов.

Стоит отметить, что подобная архитектура часто применяется при разработке приложений умеренного размера и сложности, где ее преимущества могут быть эффективно реализованы в рамках единой кодовой базы или

монолитной структуры. Однако для крупных и высоконагруженных систем обслуживание и масштабирование приложений, построенных исключительно на этой модели, без дополнительных архитектурных решений может представлять определенные сложности [12].

2.2 Проектирование структуры хранения

Одной из ключевых функциональных возможностей разрабатываемого веб-инструмента является экспорт графа интерактивного диалогового сценария в стандартизированные и легко обрабатываемые форматы данных. Это обеспечивает возможность дальнейшего использования созданных диалогов в игровых движках, системах управления контентом или других внешних приложениях. Основным форматом для экспорта и внутренней работы с данными графа является JSON, благодаря его гибкости, читаемости и широкой поддержке в веб-технологиях [13].

Центральным элементом JSON-представления является описание совокупности узлов диалога и связей между ними. Узлы представляют собой отдельные шаги, состояния или реплики в диалоге, в то время как связи определяют возможные переходы между этими узлами, обусловленные выбором пользователя или логикой сценария. Для детального описания этих концептуальных структурных элементов, а именно массива объектов узлов и массива объектов связей, были разработаны соответствующие спецификации. Подробное описание полей, входящих в структуру объекта узла, приведено в таблице 2.

Таблица 2 – Структура узла высказывания в JSON

Поле	Тип данных	Описание
id	Строка	Содержит уникальный идентификатор узла.
type	Строка	Содержит тип узла, определяющий его функциональность и внешний вид.
position	Объект	Содержит координаты x и y, определяющие положение узла на холсте редактора.

Продолжение таблицы 2

Поле	Тип данных	Описание
text	Строка	Содержит отображаемую краткую часть полного текста.
shortName	Строка	Содержит короткое имя, используемую для быстрой идентификации.
fullText	Строка	Содержит полный текст строки диалога.
options:id	Строка	Содержит уникальный идентификатор данной реакции.
options:text	Строка	Содержит текст, отображаемый пользователю для выбора этой реакции.
options:target	Строка	Содержит идентификатор узла, на который осуществляется переход при выборе данной реакции.

После того как определена структура для представления отдельных узлов диалога, включая их внутренние атрибуты и возможные варианты развития сценария через реакции, необходимо описать, каким образом эти узлы соединяются между собой, формируя общую логику и последовательность диалога. За это отвечают элементы связей, или ребер графа. Каждый такой элемент определяет направленный переход от одного узла к другому.

Детальная структура объекта, описывающего одну такую связь, представлена в таблице 3.

Таблица 3 – Структура переходов в JSON

Поле	Тип данных	Описание
id	Строка	Содержит уникальный идентификатор ребра.
source	Строка	Содержит идентификатор узла, из которого исходит ребро.
target	Строка	Содержит идентификатор узла, в который входит ребро.
sourceHandle	Строка	Содержит идентификатор точки на исходном узле, из которой выходит ребро.
targetHandle	Строка	Содержит идентификатор точки на целевом узле, в которую входит ребро.

На основе полученных результатов будет разработан и реализован алгоритм преобразования интерактивного графа, создаваемого пользователем в визуальном редакторе, в структурированный JSON-файл, полностью соответствующий описанным форматам. Такой файл будет легко поддаваться последующему автоматизированному разбору и интеграции в различные целевые системы и приложения.

2.3 Проектирование программного средства

На основе всестороннего анализа предметной области, тщательно сформулированных функциональных и нефункциональных требований, а также детально проработанных структур данных в формате JSON, было осуществлено комплексное проектирование программных средств разрабатываемого веб-инструмента. Центральным и основополагающим этапом данного процесса явилось определение основных классов системы, их атрибутов, функциональных методов и сложных взаимосвязей, формирующих архитектурное ядро приложения. Для обеспечения наглядного и структурированного представления этих фундаментальных элементов и их комплексной организации в рамках проектируемой системы традиционно прибегают к использованию диаграммы классов. Диаграмма классов предназначена для описания структуры системы на высоком уровне абстракции. Она служит мощным инструментом для визуализации и документирования архитектурных решений, демонстрируя не только сами классы как основные строительные блоки объектно-ориентированного подхода, но и их внутреннее устройство через атрибуты, определяющие состояние объектов, и операции, инкапсулирующие их поведение. Таким образом, она позволяет четко и однозначно определить ключевые компоненты будущего программного обеспечения, их внутреннюю структуру и логику их взаимодействия на концептуальном и логическом уровнях проектирования. Результаты этого этапа проектирования нашли свое отражение в диаграмме классов [14], представленной на рисунке 12.

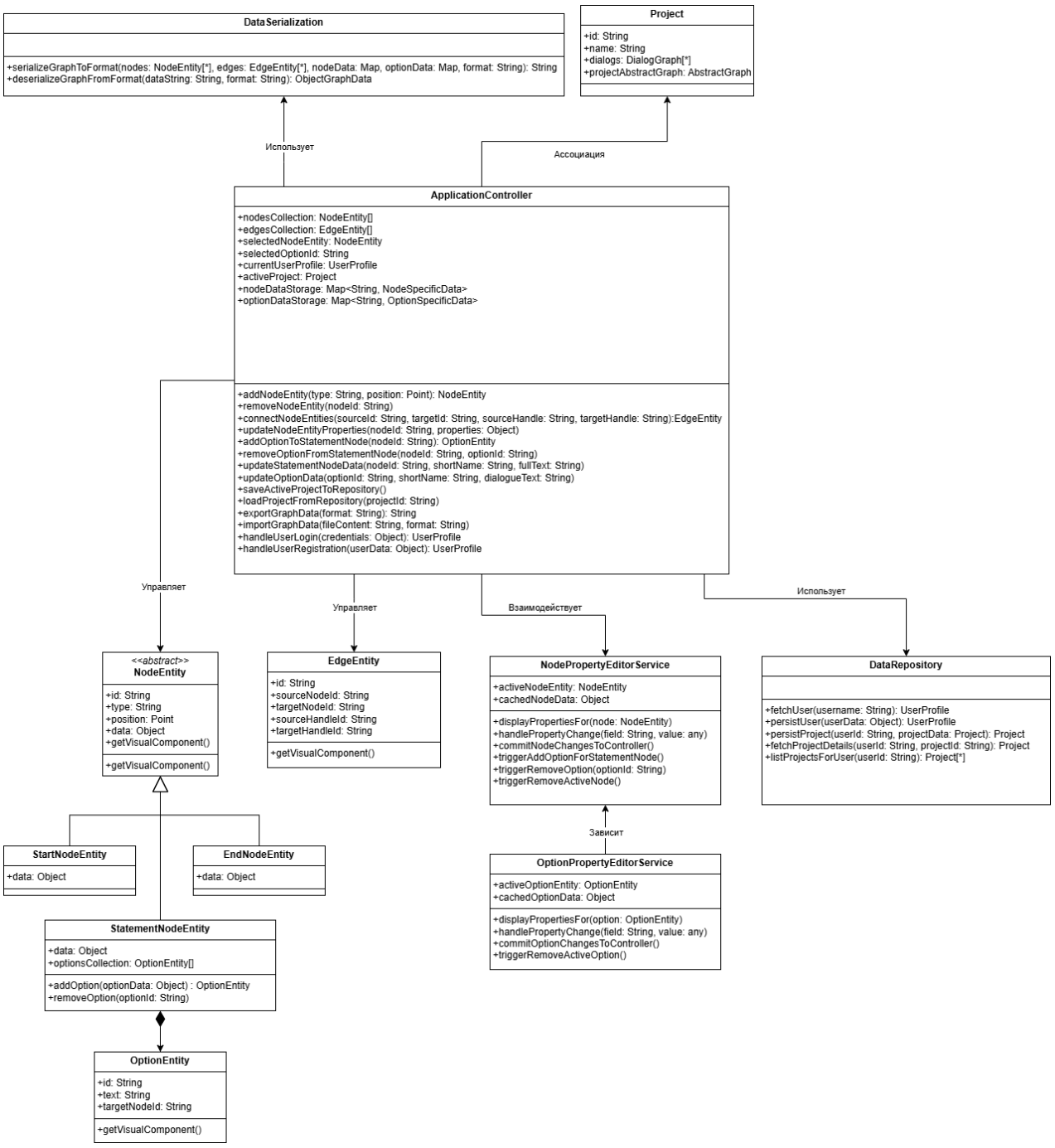

Рисунок 12 – Концептуальная диаграмма классов ключевых компонентов приложения

Для сценария, охватывающего основной функционал приложения – создание нового узла диалога – была разработана диаграмма деятельности [15], изображенная на рисунке 13. На данной диаграмме выделены две основные дорожки: «Пользователь», представляющая действия разработчика, взаимодействующего с интерфейсом, и «Система», отражающая внутренние процессы и реакции веб-инструмента на действия пользователя.

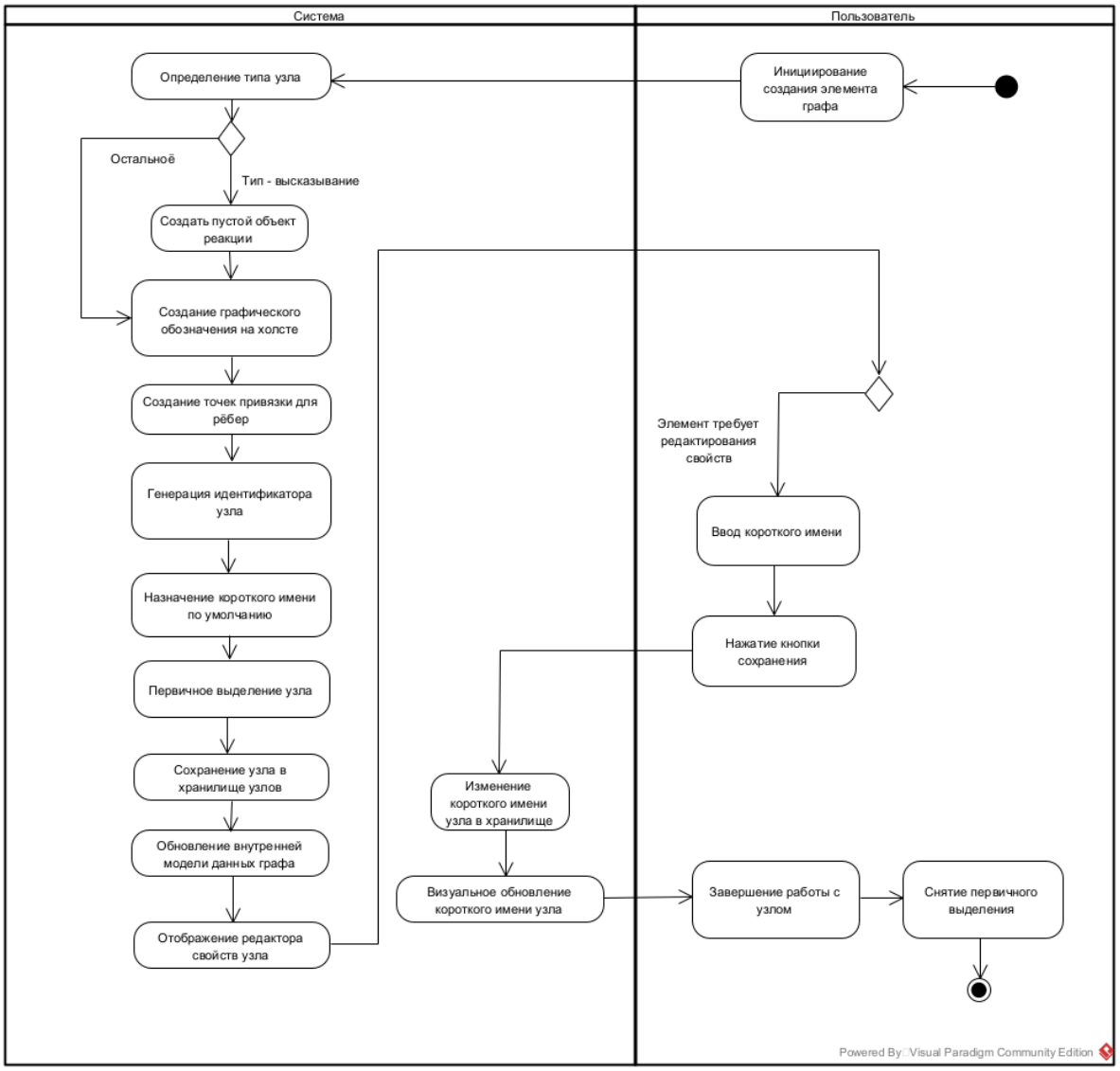

Рисунок 13 – Диаграмма деятельности для создания узла

Процесс начинается с инициации пользователем действия, например, выбора типа узла для добавления на холст. Система обрабатывает это действие: создает новый объект узла с уникальным идентификатором и начальными атрибутами, обновляет внутреннее представление графа и перерисовывает соответствующую часть пользовательского интерфейса, отображая изменения на холсте. При редактировании атрибутов узла пользователь взаимодействует с панелью редактирования, а система сохраняет внесенные изменения в хранилище объекта узла.

Для сценария экспорта созданного графа диалога в JSON, который является важной функцией для интеграции с внешними системами, была разработана диаграмма последовательности [16], представленная на рисунке 14. На данной диаграмме показано взаимодействие между основными участниками этого процесса: «Пользователь», «Бизнес-логи приложения» и «Сервис форматирования данных».

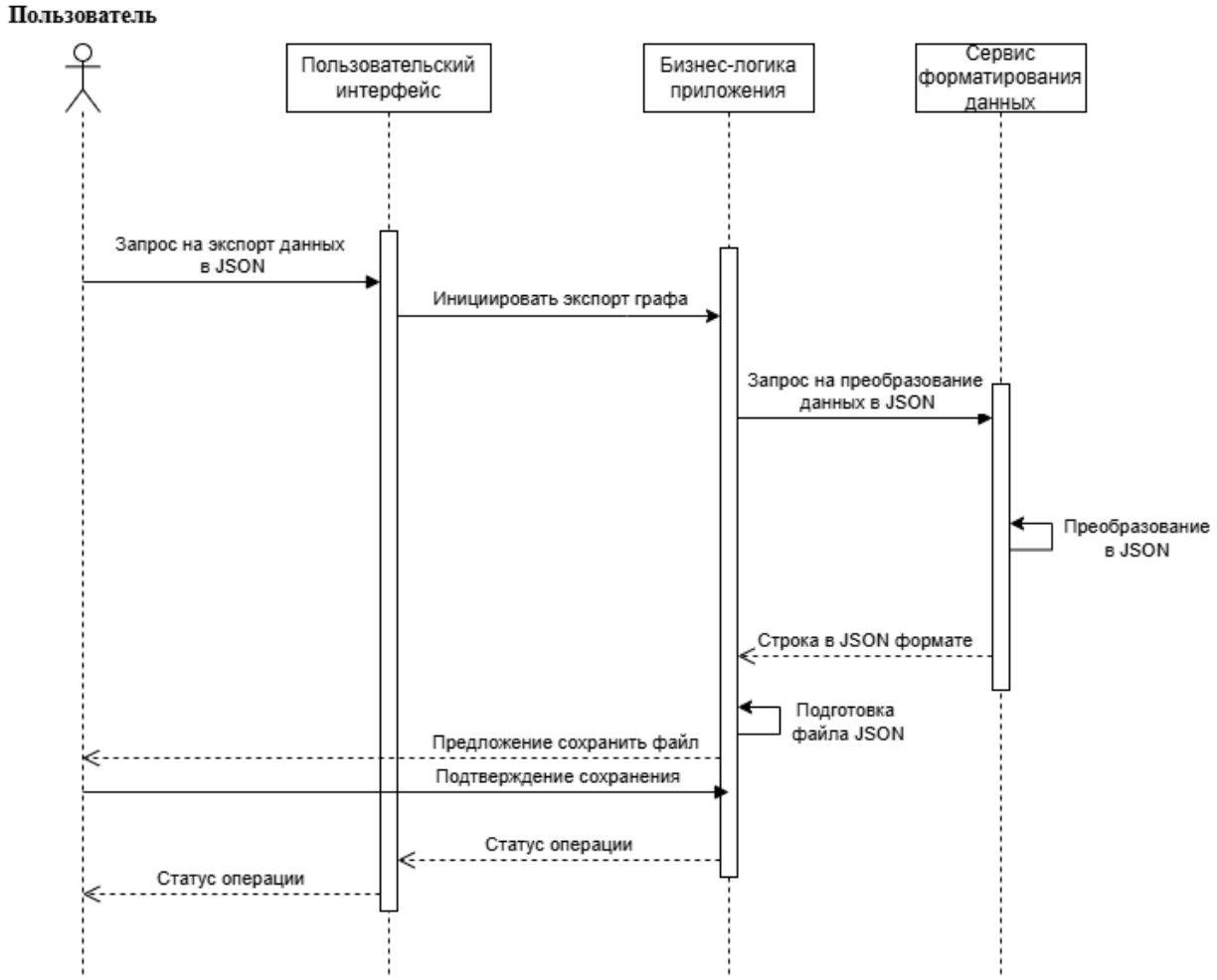

Рисунок 14 – Диаграмма последовательности экспорта в JSON

Пользователь инициирует операцию экспорта через соответствующий элемент интерфейса, выбирая желаемый формат JSON. Web-приложение собирает актуальные данные о всех узлах, связях и связанных хранилищах из своего внутреннего хранилища. Затем оно формирует JSON-объект в соответствии со спроектированной структурой данных. После формирования файла в памяти, клиентское приложение инициирует его загрузку на компьютер

пользователя, предлагая сохранить файл с соответствующим расширением.

Таким образом, в рамках данного раздела было осуществлено проектирование программных средств, что выразилось в определении структуры классов и детализации ключевых сценариев взаимодействия пользователя с системой посредством диаграмм деятельности и последовательности, формируя основу для последующей разработки.

2.4 Проектирование пользовательского интерфейса

Пользовательский интерфейс должен быть интуитивно понятен и удобен в освоении пользователем. Кроме того, интерфейс должен позволять продуктивно решать поставленные задачи по созданию и редактированию интерактивных диалогов. Для визуального проектирования эскизов интерфейса использовался инструмент Figma [17], позволяющий быстро создавать прототипы интерфейсов.

При первом запуске веб-инструмента пользователь должен попадать на главный экран, который представлен на рисунке 15. Он должен содержать минимальный набор элементов, необходимых для создания и открытия уже существующих проектов.

Рисунок 15 – Прототип главного экрана

После создания нового или открытия существующего проекта пользователь переходит на раздел управления проектом, который представлен на рисунке 16. Этот экран является центром для редактирования диалоговых сценариев внутри одного проекта.

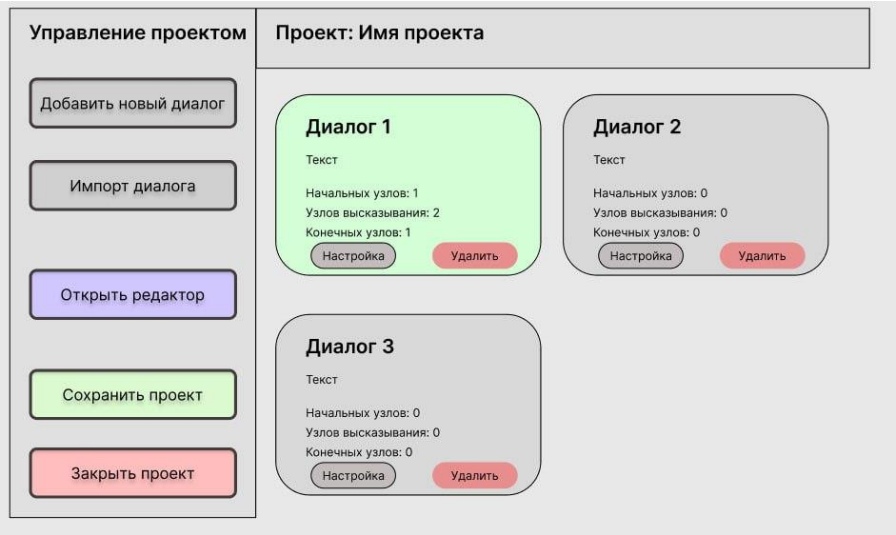

Рисунок 16 – Прототип раздела управления проектом

Основным рабочим пространством для пользователя является раздел редактора диалогов (рисунок 17). Здесь происходит непосредственное конструирование графов интерактивных сценариев.

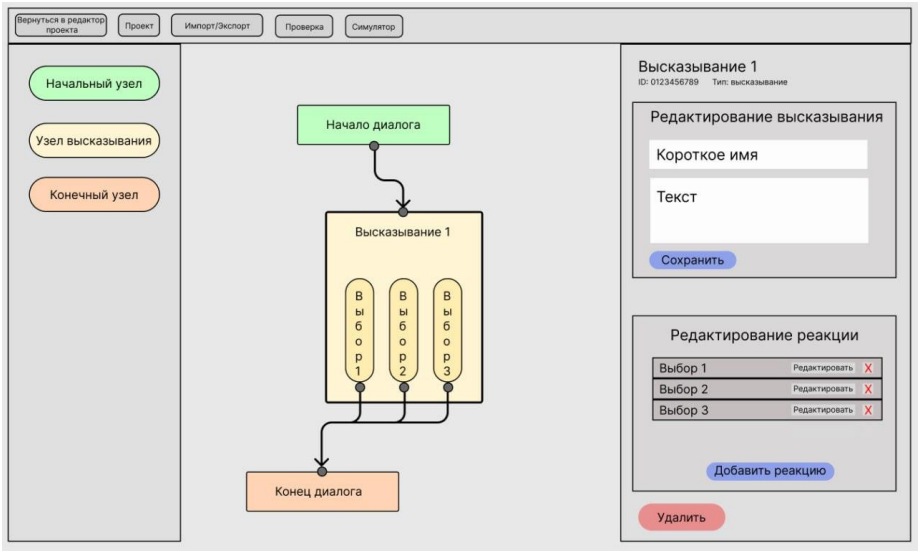

Рисунок 17 – Прототип раздела редактора диалогов

Для управления связями между отдельными диалогами в рамках одного проекта спроектирован интерфейс мета-редактора (рисунок 18). Переход в мета-редактор осуществляется с экрана управления проектом.

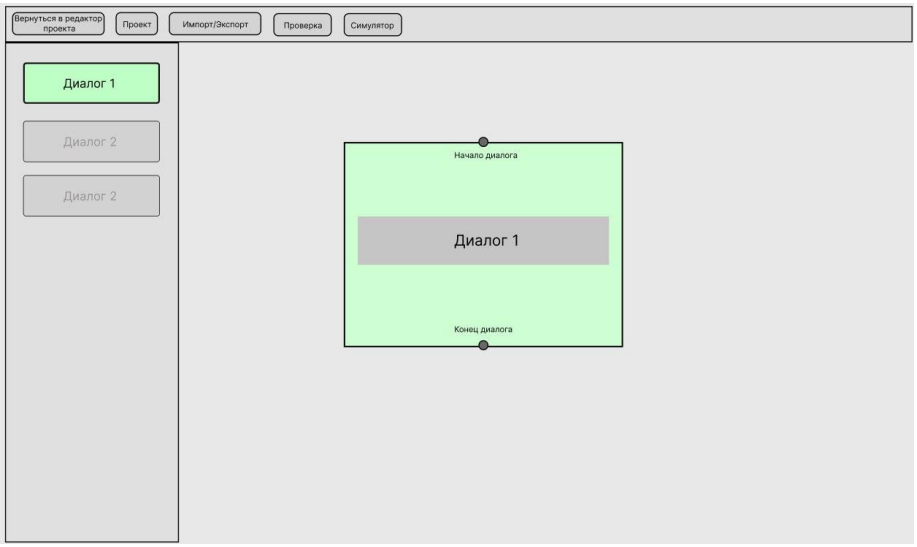

Рисунок 18 – Прототип раздела мета-редактора

Для тестирования и отладки созданных диалоговых сценариев предусмотрен режим симуляции. Интерфейс симулятора (рисунок 19) отображается в отдельном окне и позволяет пользователю пошагово проходить диалог, выбирая доступные реакции. Отображается текст текущего узла и кнопки, соответствующие возможным вариантам ответа или переходам.

Рисунок 19 – Прототип окна симулятора диалогов

Для информирования пользователя об ошибках или предупреждениях, выявленных в структуре диалога, используется окно результатов проверки (рисунок 20). В нем в структурированном виде перечисляются все обнаруженные проблемы.

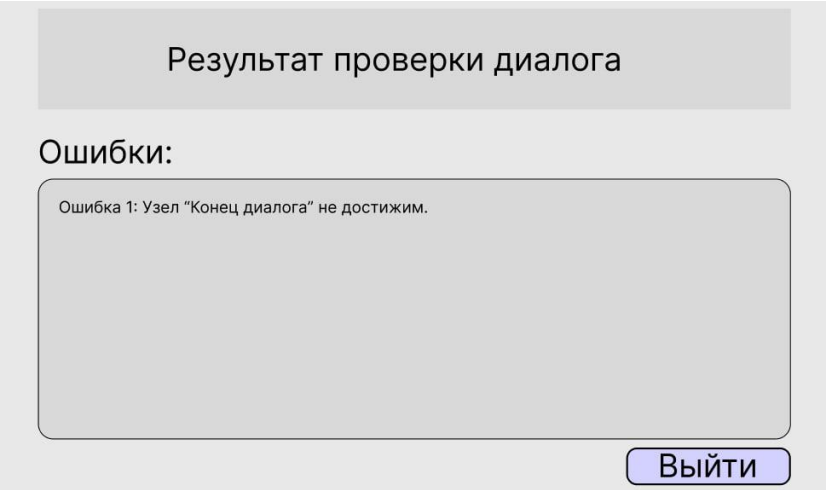

Рисунок 20 – Прототип окна результатов проверки

Таким образом, проектирование пользовательского интерфейса было направлено на создание интуитивно понятной и удобной рабочей среды, что было реализовано через разработку прототипов ключевых экранов, охватывающих все основные аспекты взаимодействия пользователя с веб-инструментом.

2.5 Планирование разработки и оценка бюджета

Планирование разработки и оценка бюджета проекта была выполнена при помощи Microsoft Project. Дата начала проекта – 8 апреля 2024 года. В результате планирования было выделено 36 задач. Задачи проекта, а также их длительность и данные о предшествующих задачах представлены на рисунке 21.

Ид.	Название задачи	Длительность	Начало	Окончание	Названия ресурсов	Предшественник
1	**Начало проекта**	**41 дней**	**Вт 08.04.25**	**Вт 03.06.25**		
2	**Постановка задачи и анализ требований**	**7 дней**	**Вт 08.04.25**	**Ср 16.04.25**		
3	Анализ предметной области	2 дней	Вт 08.04.25	Ср 09.04.25	Аналитик;Компьютер[1]	
4	Обзор аналогов	2 дней	Пт 11.04.25	Пн 14.04.25	Аналитик;Компьютер[1]	3
5	Анализ функциональных требований	2 дней	Вт 15.04.25	Ср 16.04.25	Аналитик[50%];Компьютер[1]	4
6	Анализ нефункциональных требований	2 дней	Вт 15.04.25	Ср 16.04.25	Аналитик[50%];Компьютер[1]	4
7	Завершение постановки задачи и анализа требований	0 дней	Ср 16.04.25	Ср 16.04.25		6
8	**Проектирование программных средств**	**12 дней**	**Чт 17.04.25**	**Пт 02.05.25**		
9	Проектирование архитектуры приложения	3 дней	Чт 17.04.25	Пн 21.04.25	Аналитик;Компьютер[1];Программист;Проектировщик	7
10	Проектирование структуры хранения	2 дней	Вт 22.04.25	Ср 23.04.25	Компьютер[1];Проектировщик	9
11	Проектирование приложения	4 дней	Чт 24.04.25	Вт 29.04.25	Компьютер[1];Программист;Проектировщик	10
12	Проектирование пользовательского интерфейса	3 дней	Ср 30.04.25	Пт 02.05.25	Дизайнер;Компьютер[1];Программист;Проектировщик	11
13	Завершение пректирования программных средств	0 дней	Пт 02.05.25	Пт 02.05.25		12
14	**Реализация программных средств**	**12 дней**	**Пн 05.05.25**	**Вт 20.05.25**		
15	Реализация слоя данных приложения	3 дней	Пн 05.05.25	Ср 07.05.25	Программист;Компьютер[1]	13
16	Реализация слоя бизнес-логики приложения	5 дней	Чт 08.05.25	Ср 14.05.25	Программист;Компьютер[1]	15
17	Реализация слоя представления приложения	4 дней	Чт 15.05.25	Вт 20.05.25	Дизайнер;Программист;Компьютер[1]	16
18	Завершение реализации программных средств	0 дней	Вт 20.05.25	Вт 20.05.25		17
19	**Тестирование**	**8 дней**	**Ср 21.05.25**	**Пт 30.05.25**		
20	Функциональное тестирование	3 дней	Ср 21.05.25	Пт 23.05.25	Компьютер[1];Тестировщик	18
21	Анализ сложности кода	3 дней	Пт 23.05.25	Ср 28.05.25	Тестировщик;Компьютер[1]	20
22	Автоматизированное тестирование	2 дней	Чт 29.05.25	Пт 30.05.25	Программист;Тестировщик;Компьютер[1]	21
23	Подготовка документации	0 дней	Чт 29.05.25	Чт 29.05.25	Бумага[1];Принтер[1];Компьютер[1];Аналитик	22
24	Оформление и печать документации	3 дней	Пт 30.05.25	Вт 03.06.25	Бумага[1];Принтер[1];Компьютер[1];Аналитик	23
25	Завершение проекта	0 дней	Вт 03.06.25	Вт 03.06.25		24

Рисунок 21 – Список задач проекта

Список ресурсов, используемых при выполнении задач представлен на рисунке 22.

	Название ресурса	Тип	Стандартная ставка	Начисление	Базовый календарь	Ставка сверхурочны	Макс. единиц	Затраты на использ.
1	Аналитик	Трудовой	238,00 ₽/ч	Пропорциональное	Стандартный	357,00 ₽/ч	100%	0,00 ₽
2	Программист	Трудовой	300,00 ₽/ч	Пропорциональное	Стандартный	450,00 ₽/ч	100%	0,00 ₽
3	Тестировщик	Трудовой	286,00 ₽/ч	Пропорциональное	Стандартный	429,00 ₽/ч	100%	0,00 ₽
4	Дизайнер	Трудовой	286,00 ₽/ч	Пропорциональное	Стандартный	429,00 ₽/ч	100%	0,00 ₽
5	Проектировщик	Трудовой	290,00 ₽/ч	Пропорциональное	Стандартный	435,00 ₽/ч	100%	0,00 ₽
6	Компьютер	Материальный	40 000,00 ₽	В начале				0,00 ₽
7	Бумага	Материальный	500,00 ₽	В начале				0,00 ₽
8	Принтер	Материальный	5 000,00 ₽	В начале				0,00 ₽

Рисунок 22 – Список ресурсов

В результате выделения задач и ресурсов была построена диаграмма Ганта (рисунок 23). На ней представлены продолжительность задач и взаимосвязи их друг с другом [18].

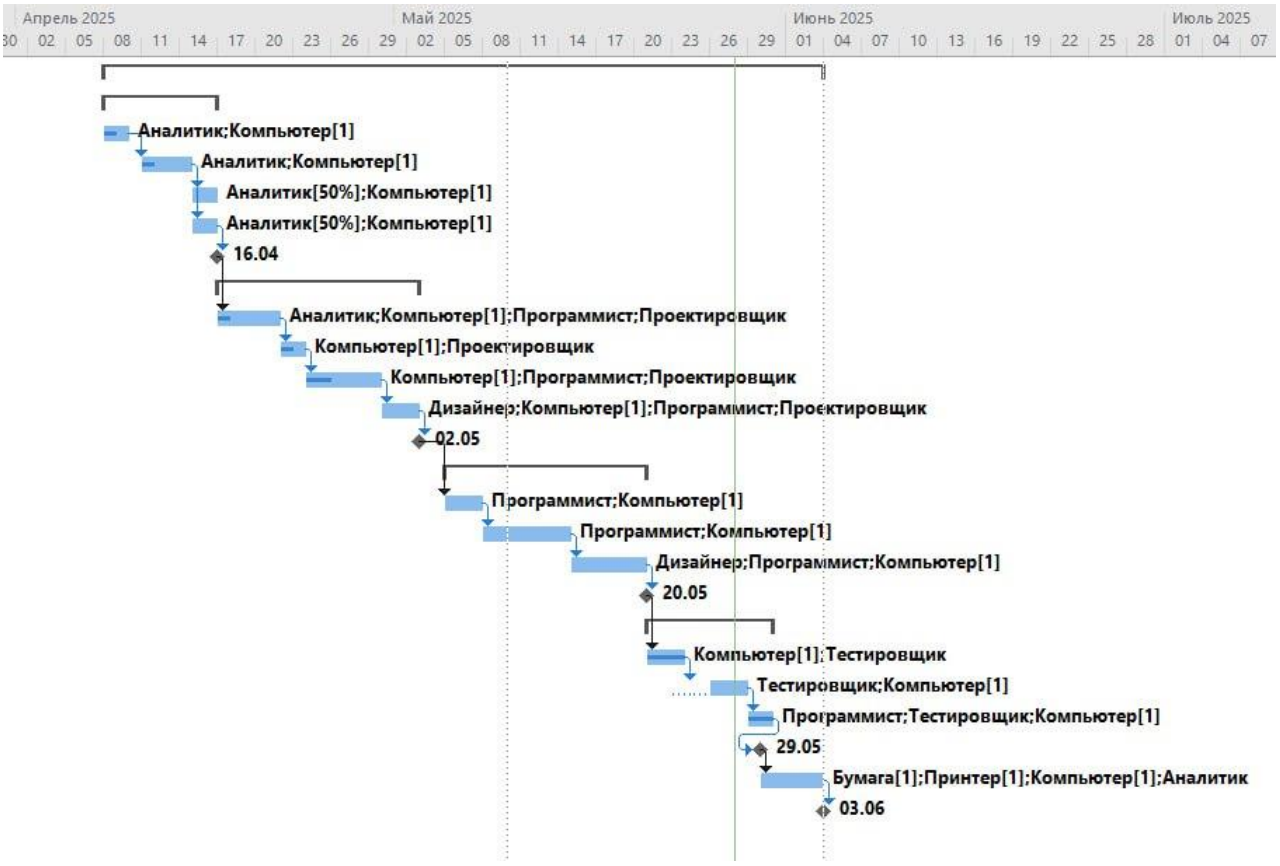

Рисунок 23 – Диаграмма Ганта для задач проекта

На рисунке 24 представлена сводная информация по проекту, содержащая информацию о длительности и бюджете.

	Начало	Окончание
Текущее	Вт 08.04.25	Вт 03.06.25
Базовое	НД	НД
Фактическое	Вт 08.04.25	НД
Отклонение	0д	0д

	Длительность	Трудозатраты	Затраты
Текущие	41д	480,17ч	146 952,00 ₽
Базовые	0д	0ч	0,00 ₽
Фактические	10,49д	136ч	43 932,00 ₽
Оставшиеся	30,51д	344,17ч	103 020,00 ₽

Рисунок 24 – Сводная информация по проекту

Таким образом, разработка проекта составит 41 полный рабочий день с 08.04.25 по 03.06.25. Трудозатраты составят 480,17 человеко-часов. Оценка бюджета проекта составляет 146 952,00 рублей.

3 Реализация программных средств веб-инструмента для создания и редактирования интерактивных диалогов в игровых приложениях

Слой данных в веб-инструменте отвечает за структурированное представление, хранение и управление информацией о проектах, диалогах, их компонентах и сопутствующих метаданных. Реализация слоя данных сосредоточена на клиентской стороне, используя возможности браузера для работы с файлами (импорт и экспорт) и управление состоянием приложения.

3.1 Реализация слоя данных приложения

Основой управления данными на уровне всего проекта является состояние currentProject в главном компоненте DialogSystemWrapper.jsx. Этот объект агрегирует всю информацию о текущем открытом проекте, включая его уникальный идентификатор, имя, массив всех диалогов, структуру мета-графа.

При создании нового проекта используется функция createNewProjectData, которая создает проект, в котором можно создавать отдельные диалоги, а затем связывать их между собой. Загрузка существующего проекта из файла осуществляется функцией loadProjectFromFile, которая парсит файл. Сохранение всего проекта в файл реализуется функцией saveProjectToFile, которая преобразует актуальное состояние currentProject в JSON и инициирует загрузку файла.

Обновление данных проекта в состоянии приложения DialogSystemWrapper.jsx происходит через функцию updateCurrentProjectData. Эта функция не только обновляет состояние, но и пересчитывает признак полноты для каждого диалога, а также может инициировать автоматическое сохранение проекта в файл.

Ключевой функцией для обмена данными отдельных диалоговых сценариев является их экспорт и импорт в формате JSON, структура которого была спроектирована в разделе 2.2 (таблицы 2 и 3). Эта функциональность реализована в компоненте App.js, который отвечает за редактирование графа

диалога.

Экспорт графа диалога в JSON файл осуществляется функцией exportGraphToJson. Она собирает текущее имя диалога, массивы узлов и связей, а также хранилища данных для высказываний и реакци в единый объект. Этот объект затем конвертируется в JSON-строку и предлагается пользователю для сохранения. Поле statementDataStorage содержит объекты, где ключ – id узла высказывания, а значение – объект `{shortName, fullText}`. Поле optionDataStorage содержит объекты, где ключ – id реакции, а значение – объект `{shortName, dialogueText}`. Листинг 1 демонстрирует фрагмент этой функции.

Листинг 1 – Фрагмент функции экспорта графа диалога в JSON

```
const exportGraphToJson = useCallback(() => {
  const fileName = promptForFileName(currentName, 'json');
  if (!fileName) return;
  const graphData = {
    name: currentName,
    nodes: nodes,
    edges: edges,
    statementDataStorage: statementData,
    optionDataStorage: optionData
  };
  const jsonString = JSON.stringify(graphData, null, 2);
  const blob = new Blob([jsonString], { type: 'application/json'
});
  setShowExportMenu(false);
}, [currentName, nodes, edges, statementData, optionData]);
```

Импорт графа диалога из JSON файла реализован в функции importGraphFromJson. Она читает содержимое файла, парсит JSON и обновляет состояние компонента App.js (имя диалога, узлы, связи и хранилища данных). Листинг 2 показывает фрагмент этой функции.

Листинг 2 – Фрагмент функции импорта графа диалога из JSON

```
const importGraphFromJson = useCallback((event) => {
  const file = event.target.files[0];
  if (!file) return;
  const reader = new FileReader();
  reader.onload = (e) => {
    try {
```

```
      const importedData = JSON.parse(e.target.result);
      setCurrentName(importedData.name ||
file.name.replace(/\.json$/i, ''));
      setNodes(deserializeNodes(importedData.nodes || []));
      setEdges(deserializeEdges(importedData.edges || []));
      setStatementData(importedData.statementDataStorage || {});
      setOptionData(importedData.optionDataStorage || {});
      setSelectedNode(null);
      resetAllErrorHighlighting();
      alert("Данные из .json импортированы...");
    } catch (err) {…}
  };
  reader.readAsText(file);
  event.target.value = null;
  setShowExportMenu(false);
}, [setCurrentName, setNodes, setEdges, setStatementData,
setOptionData, resetAllErrorHighlighting]);
```

Для наглядного представления взаимодействия основных компонентов, участвующих в управлении данными, на рисунке 25 приведена упрощенная диаграмма классов.

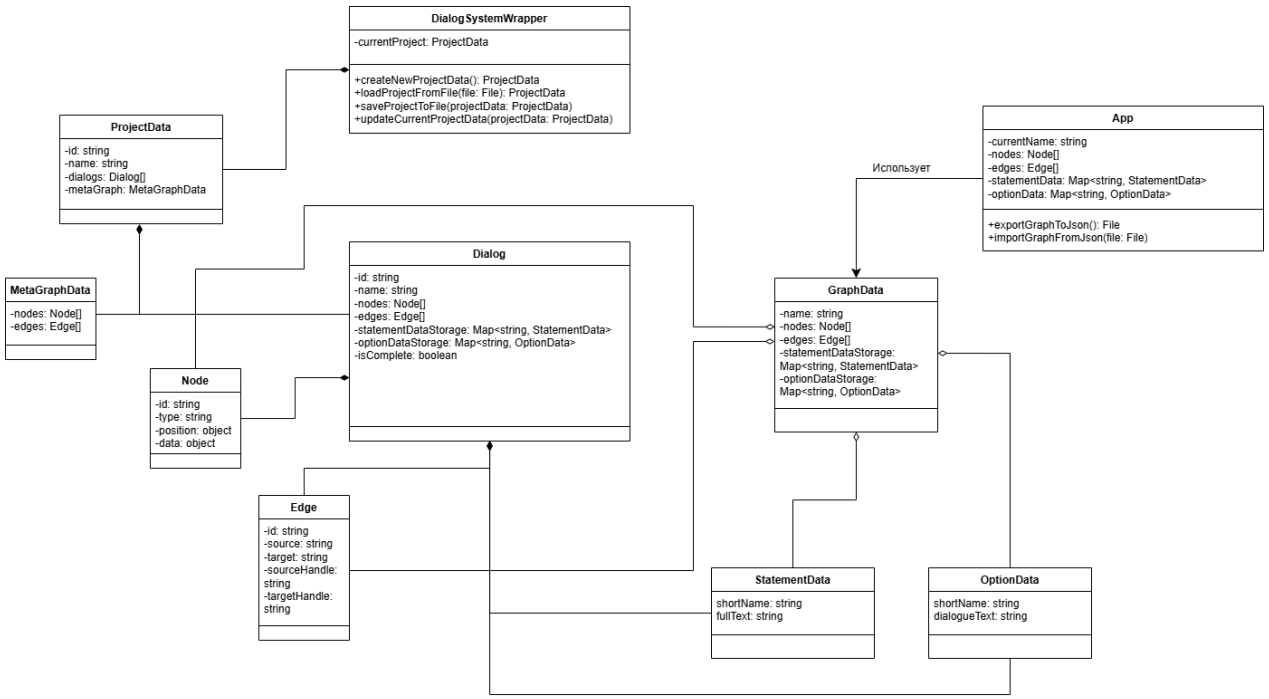

Рисунок 25 – Диаграмма классов, участвующих в управлении данными

Реализация слоя данных обеспечивает как управление проектом в целом, так и работу с отдельными диалоговыми графами, включая их импорт и экспорт в спроектированном JSON-формате. Данные хранятся и передаются между компонентами в четко определенных структурах, с использованием

утилитарных функций, что обеспечивает возможность обмена данными с другими системами.

3.2 Реализация слоя бизнес-логики приложения

Слой бизнес-логики в разрабатываемом веб-инструменте отвечает за обработку пользовательских действий, управление состоянием графа, валидацию данных и координацию взаимодействия между слоем представления и слоем данных. Основная бизнес-логика сосредоточена в компоненте App.js, который выступает в роли контроллера для редактора графов, а также в компоненте DialogSystemWrapper.jsx для управления проектом в целом и в ProjectView.jsx для управления списком диалогов.

Компонент App.js использует хуки useNodesState и useEdgesState из библиотеки React Flow для управления массивами узлов и связей. Эти хуки предоставляют функции для обновления состояния, которые автоматически вызывают перерисовку графа.

При перетаскивании элемента из боковой панели на холст, срабатывает обработчик onDrop. Он получает тип перетаскиваемого узла, генерирует для него уникальный id с помощью универсального уникального идентификаторв (UUID) версии 4: uuidv4(), определяет его начальные координаты и структуру данных в соответствии с типом узла. Например, для узла statement создается объект с пустым массивом options и начальными значениями shortName и fullText. Новый узел добавляется в массив nodes с помощью функции setNodes. Фрагмент кода обработчика onDrop представлен в листинге 3.

Листинг 3 – Фрагмент обработчика onDrop для добавления узлов

```
const onDrop = useCallback((event) => {
  event.preventDefault();
  if (!reactFlowInstance || !reactFlowWrapper.current) return;
  const position = reactFlowInstance.project({…});
  const newNodeId = uuidv4();
  let newNode;
  const commonData = { isError: false, isWarning: false };
  switch (nodeInfo.type) {
    case 'start':
      newNode = { id: newNodeId, type: 'start', position, data: {
```

```
label: 'Вход', ...commonData }, style: { zIndex: 10 } };
      break;
    case 'end':
      newNode = { id: newNodeId, type: 'end', position, data: {
label: 'Выход', ...commonData }, style: { zIndex: 10 } };
      break;
    case 'statement':
      newNode = { id: newNodeId, type: 'statement', position,
data: { label: 'Высказывание', shortName: 'Высказывание',
fullText: '', options: [], ...commonData }, style: { zIndex: 10 }
};
      setStatementData(prev => ({...prev, [newNodeId]: {shortName:
'Высказывание', fullText: ''}}));
      break;
    case 'dialogRef':
      newNode = { id: newNodeId, type: 'dialogRef', position,
data: {
          label: nodeInfo.label || 'Ссылка', dialogId:
nodeInfo.dialogId,
          entryPoints: nodeInfo.entryPoints || [],
          exitPoints: nodeInfo.exitPoints || [],
          ...commonData,
      }, style: { zIndex: 10 }};
      break;
    default: console.warn("Unknown node type:", nodeInfo.type);
return;
  }
  setNodes((nds) => nds.concat(newNode));
  resetAllErrorHighlighting();
}, [reactFlowInstance, setNodes, setStatementData,
resetAllErrorHighlighting]);
```

Удаление узла: Функция deleteNode удаляет указанный узел из массива nodes и все связанные с ним ребра из массива edges. Также удаляются связанные данные из statementData и optionData. Если удаляемый узел был выбран, выбор сбрасывается. Код функции deleteNode приведен в листинге 4.

Листинг 4 – Реализация функции deleteNode

```
const deleteNode = useCallback((nodeIdToDelete) => {
  const nodeToDelete = nodes.find(n => n.id === nodeIdToDelete);
  setNodes(nds => nds.filter(n => n.id !== nodeIdToDelete));
  setEdges(eds => eds.filter(edge => edge.source !==
nodeIdToDelete && edge.target !== nodeIdToDelete));
  setNodes(nds => nds.map(n => {
    if (n.type === 'statement' && n.data.options) {
      const newOptions = n.data.options.map(opt =>
        opt.target === nodeIdToDelete ? { ...opt, target: '' } :
opt
      );
```

```
      if (JSON.stringify(n.data.options) !==
JSON.stringify(newOptions)) {
        return { ...n, data: { ...n.data, options: newOptions } };
      }
    }
    return n;
  }));
}, [nodes, selectedNode, setNodes, setEdges, setOptionData,
setStatementData, setValidationIssues]);
```

Создание связи: Обработчик onConnect вызывается React Flow при успешном соединении двух Handle. Он создает новый объект ребра с уникальным id, sourceHandle, targetHandle и добавляет его в массив edges с помощью setEdges. Также обновляются данные узла-источника, если это узел statement или start. Код обработчика onConnect представлен в листинге 5.

Листинг 5 – Реализация обработчика onConnect

```
const onConnect = useCallback((params) => {
  const sourceHandle = params.sourceHandle || 'source-default-
${params.source}';
  const targetHandle = params.targetHandle || 'target-default-
${params.target}';
  const newEdge = { id: 'edge-${uuidv4()}', ...params,
sourceHandle, targetHandle, type: 'smoothstep', style: { zIndex:
100, strokeWidth: 1.5 } };
  setEdges((eds) => addEdge(newEdge, eds));
  setNodes(nds => nds.map(n => {
      return n;
  }));
}, [validationIssues, setEdges, setNodes]);
```

Слой бизнес-логики эффективно управляет состоянием и поведением приложения. В App.js реализованы ключевые операции по управлению графом диалога, включая добавление, удаление, связывание элементов и валидацию. DialogSystemWrapper.jsx и ProjectView.jsx обрабатывают логику на уровне проекта и списка диалогов. Симулятор диалогов инкапсулирует сложную логику пошагового выполнения диалоговых сценариев, включая обработку вложенных диалогов в мета-графе. Взаимодействие между компонентами построено на передаче данных через пропсы и вызове колбэков, что соответствует принципам React. Приложение Б содержит исходный код программного средства.

3.3 Разработка слоя представления приложения

При разработке графического интерфейса веб-инструмента использовался фреймворк React совместно с библиотекой React Flow для визуализации графовых структур. Такой подход позволил создать интерактивное одностраничное приложение (SPA), обеспечивающее удобное взаимодействие пользователя с системой. Для отображения информации, управления проектами, диалогами и их элементами были разработаны различные экраны и компоненты.

При первом входе в приложение пользователь попадает на главный экран (рисунок 26). Данный экран встречает пользователя названием веб-инструмента и предлагает два основных действия: «Создать новый проект» для начала работы с чистого листа и «Открыть проект» для загрузки ранее сохраненного проекта из файла.

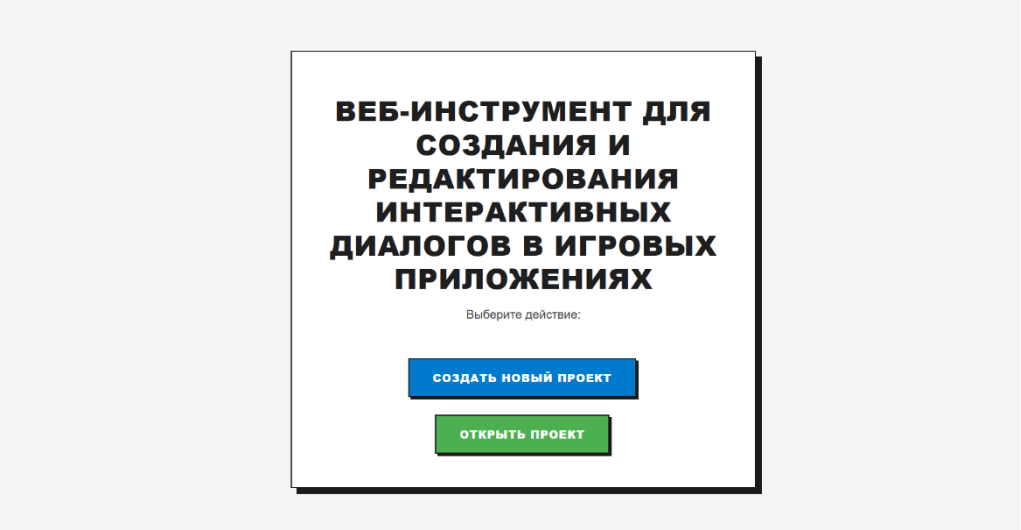

Рисунок 26 – Интерфейс главного меню приложения

После создания нового или открытия существующего проекта пользователь переходит в раздел управления проектом (рисунок 27). Этот экран является центральным узлом для работы с диалогами внутри проекта. Слева расположена боковая панель, предоставляющая основные команды: «Добавить новый диалог» для создания пустого диалогового сценария в текущем проекте, «Импорт диалога из файла» для добавления диалога из файла, «Открыть Мета-Редактор» для перехода к редактированию связей между диалогами, «Сохранить

весь проект» для записи всех изменений в файл и «Закрыть проект» для возврата в главное меню.

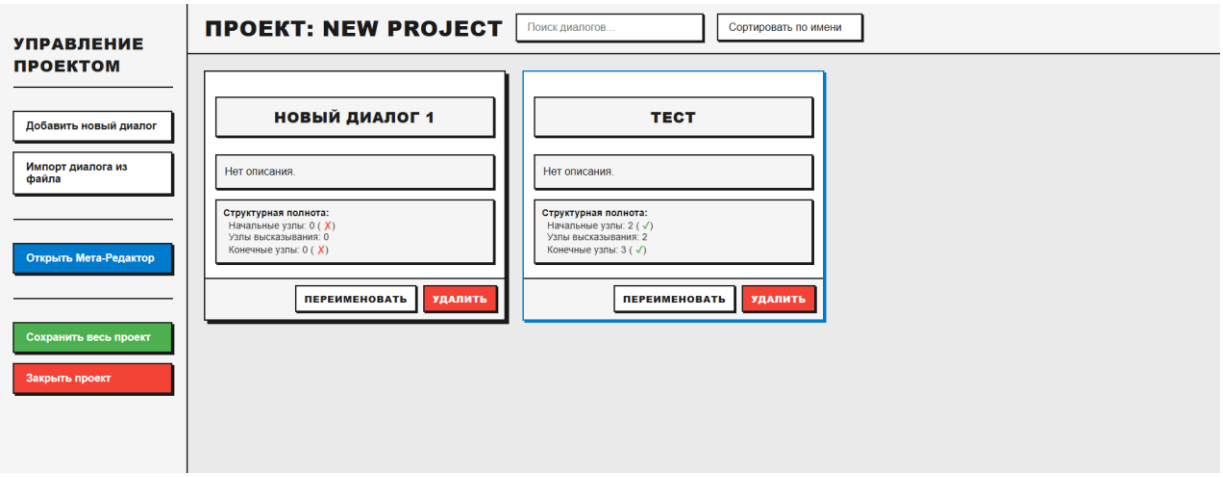

Рисунок 27 – Интерфейс раздела управления проектами

Сами диалоги представлены в виде набора карточек, на карточке отображается имя диалога, его краткое описание, информация о структурной полноте (наличие начальных, конечных узлов). Пользователь может переименовать диалог или удалить его прямо с карточки. Двойной клик по карточке диалога или ее выбор с последующим подтверждением открывает данный диалог для редактирования.

При открытии диалога для редактирования открывается раздел для интерактивного конструирования графов (рисунок 28).

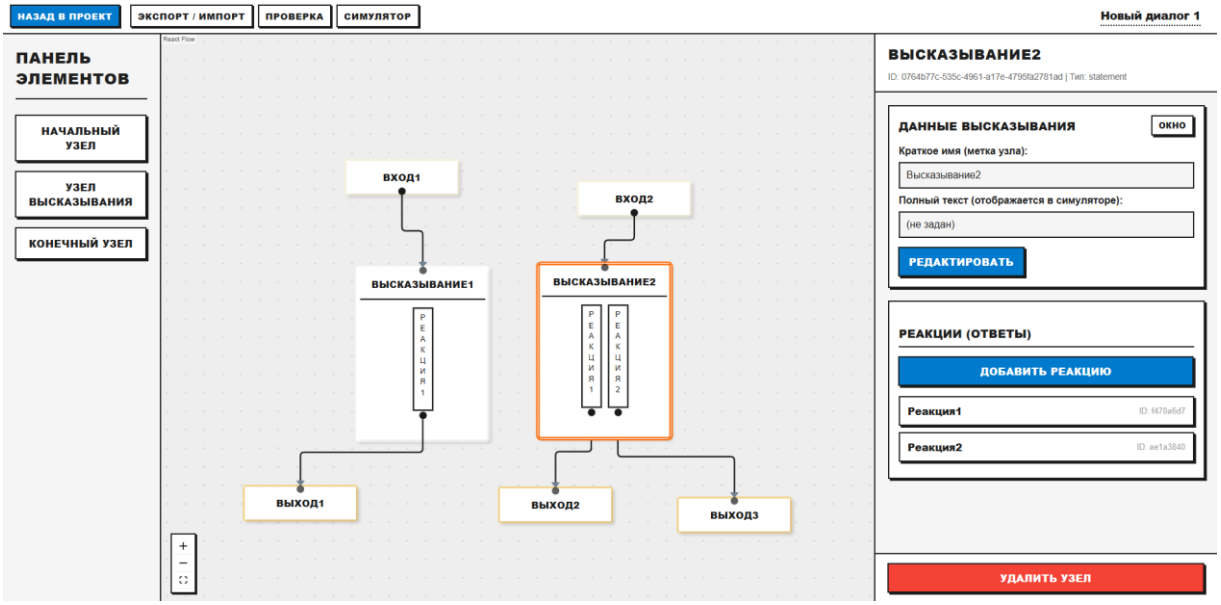

Рисунок 28 – Интерфейс редактора диалога

В верхней части этого экрана расположена панель инструментов. Она содержит кнопки и выпадающие меню для сохранения, экспорта, импорта текущего графа в JSON, а также экспорт в графические форматы PNG и JPG. Здесь же находятся кнопки «Проверка» для запуска валидации текущего графа и «Симулятор» для его интерактивного тестирования. В правой части панели отображается имя редактируемого диалога или мета-графа.

Слева от холста редактора располагается боковая панель элементов. Эти элементы можно перетащить на холст для создания узла.

Основное пространство редактора занимает холст, реализованный с помощью библиотеки React Flow. На нем пользователь визуально конструирует граф, добавляя узлы перетаскиванием из боковой панели, перемещая их и соединяя их точки связи для создания ребер. Холст поддерживает масштабирование и панорамирование. Для удобства навигации присутствуют стандартные элементы управления React Flow.

При выборе узла на холсте справа обновляется панель редактирования. Она позволяет детально редактировать атрибуты выбранного узла. Для узлов «Начальный узел» и «Конечный узел» доступно редактирование их текстовой метки. Для «Узла высказывания» можно задать и полный текст. Важной частью редактора для узла высказывания является секция управления реакциями: кнопка «Добавить реакцию» и список существующих реакций. Для удобства работы с объемным текстом (например, полный текст высказывания или реакции) предусмотрена возможность открытия отдельного окна редактирования.

Для проверки и отладки созданных диалогов предусмотрен симулятор (рисунок 29). При его запуске отображается текст или информация текущего узла, а также список доступных реакций или переходов в виде кнопок. Пользователь может выбирать опции, перемещаясь по диалогу. Симулятор поддерживает историю переходов с возможностью шага назад и корректно обрабатывает переходы. Ошибки, выявленные в ходе валидации графа, отображаются в отдельном модальном окне (рисунок 30).

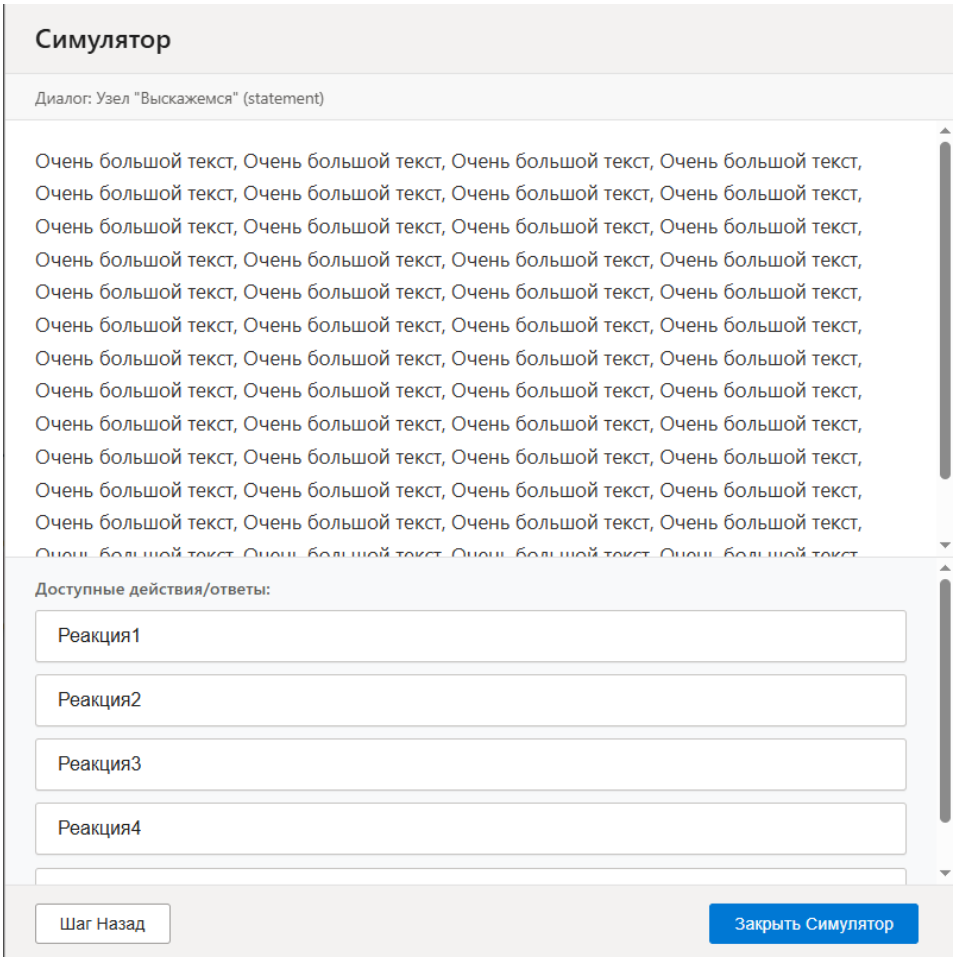

Рисунок 29 – Интерфейс симулятора диалога

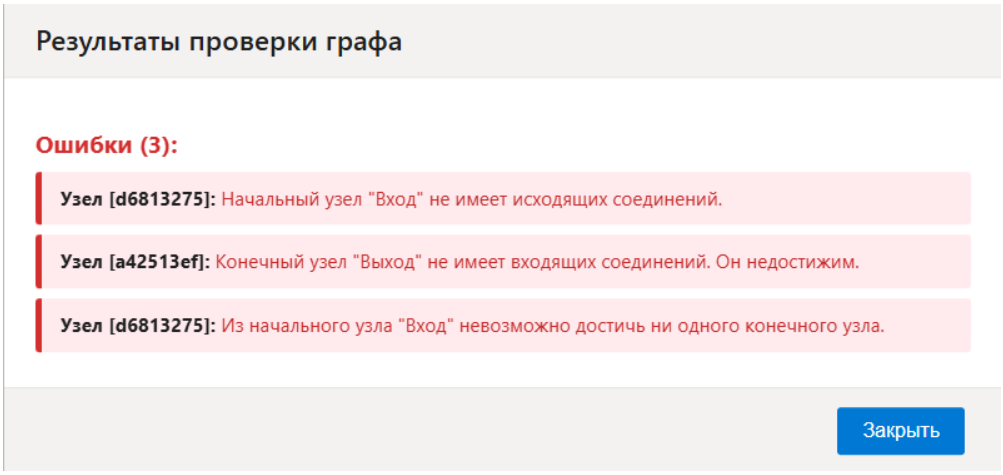

Рисунок 30 – Интерфейс проверки правильности составления редактора

Помимо редактирования отдельных диалоговых сценариев, веб-инструмент предоставляет функциональность метаредактора, предназначенного для управления связями между отдельными диалогами в рамках одного проекта. Интерфейс метаредактора представлен на рисунке 31. Переход в метаредактор

осуществляется из раздела управления.

Боковая панель слева в метаредакторе отображает список всех завершенных диалогов текущего проекта. Завершенность диалога определяется на основе его структурной полноты, наличия начальных и конечных узлов, а также связности всех элементов. Только завершенные диалоги могут быть добавлены на холст метаредактора путем перетаскивания.

Пользователь может соединять узлы, создавая связи между любой точкой входа одного диалога и точкой выхода другого, тем самым выстраивая последовательность прохождения группы диалогов. Это позволяет проектировать сложные многоэтапные сценарии, где каждый этап представлен отдельным, независимо разработанным диалогом.

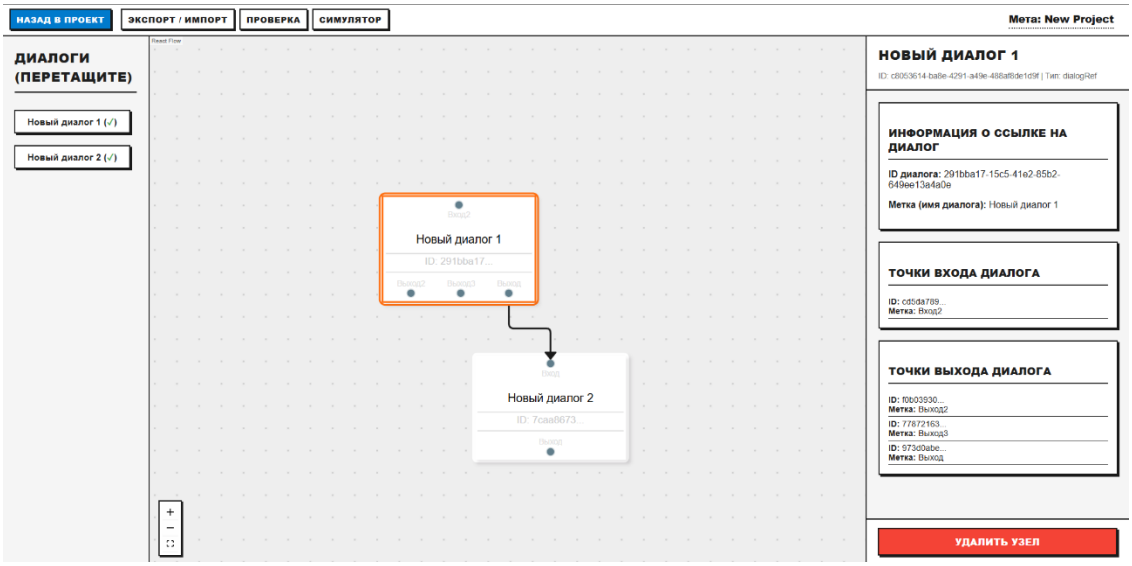

Рисунок 31 – Интерфейс мета-редактора

Таким образом, слой представления веб-инструмента предоставляет пользователю полный набор визуальных средств для эффективного создания, редактирования, управления и тестирования интерактивных диалоговых сценариев. Компонентный подход фреймворка React позволил создать гибкий и расширяемый интерфейс.

Была успешно осуществлена практическая реализация веб-инструмента для создания и редактирования интерактивных диалогов. Была реализована трехуровневая архитектура приложения: слой данных обеспечил управление

проектами и диалогами, включая их импорт и экспорт в форматы JSON; слой бизнес-логики инкапсулировал операции по манипулированию элементами графа, их валидации и логику работы симулятора; слой представления предоставил интуитивно понятный пользовательский интерфейс для всех ключевых функций, включая визуальное конструирование диалогов, управление проектами и тестирование сценариев. Взаимодействие компонентов построено на принципах React, обеспечивая управляемый поток данных. Итогом работы стал функциональный прототип веб-инструмента, соответствующий поставленным требованиям и готовый к дальнейшему тестированию и развитию.

4 Тестирование модуля создания узлов

4.1 Тестирование создания узлов методом чёрного ящика

Тестирование методом чёрного ящика определяется как методика тестирования, при которой функциональность тестируемого приложения, тестируется без учета внутренней структуры кода, деталей реализации и знания внутренних путей программного обеспечения. Этот тип тестирования полностью основан на требованиях и спецификациях программного обеспечения. В тестирование методом чёрного ящика фокусируются на входах и выходах программной системы, не заботясь о внутренних знаниях программ. Вышеупомянутый чёрного ящик может быть любой программной системой, которую нужно протестировать. В тестировании методом чёрного ящика можно протестировать эти приложения, просто сосредоточившись на входах и выходах, не зная их внутренней реализации кода [19].

4.1.1 Классы эквивалентности

При создании узла, требуется указать ему краткое имя, которое помогает визуально ориентироваться на графическом полотне Поле «Краткое имя реакции» должно содержать 2 – 8 символов: цифры, латинские буквы, кириллицу, специальные символы. Должно начинаться с буквы или цифры. В таблице 4 приведены классы эквивалентности для краткого имени.

Таблица 4 – Классы эквивалентности для краткого имени

	Правильные КЭ	Неправильные КЭ
Длина краткого имени реакции	2 – 8 (1)	0 – 1(2) 9 – ∞(3)
Структура	Начинается с буквы (4) Начинается с цифры(5)	Начинается со специального символа (6)
Содержание	Буквы Латинские (7) Киррилицу (8) Цифры (9) Специальные символы (10)	Остальное (11)

Графическое представление классов эквивалентности изображено на рисунках 32, 33.

Рисунок 32 – Класс эквивалентности (длина текста высказывания)

Рисунок 33 – Класс эквивалентности (содержание)

Таблица 5 – Тест-кейсы для поля «краткое имя»

Номер теста	Классы Эквивалентности	Входные данные	Выходные данные
1	1,4,8,10	Привет!)	Успешно
2	2,4,7	Ы	Ошибка: Меньше 2х символов
3	3,5,9	123456789101112	Ошибка: Больше 8х символов
4	1,6,9,10	№1	Ошибка: Начинается со специального символа
5	1,11	文7676	Ошибка: Содержит неподдерживаемый символ

Таким образом, были построены классы эквивалентности для тестирования краткого имени.

4.1.2 Функциональная диаграмма и таблица решений

Чтобы создать диалог, пользователь должен иметь возможность создавать и редактировать диалоговые графы. Для этого он должен добавить начальный и конечный узел, между которыми размещается узел высказывания. Начальный узел может быть только один. Начальный и конечный узел могут содержать

только краткое имя длиной 2 – 8 символа. Краткое имя состоит из букв, цифр, специальных символов. Пользователь не может продублировать краткое имя уже существующего узла. Таким образом была сформирована таблица 6.

Таблица 6 – Набор причин и следствий

Причина	Следствие
П1	Начальный узел присутствует
П2	Конечный узел присутствует
П3	Узел высказывания присутствует
П4	Корректная длина краткого имени начального узла
П5	Корректная длина краткого имени конечного узла
П6	Корректная длина краткого имени узла высказывания
П7	Краткое имя начального узла состоит из корректных символов
П8	Краткое имя конечного узла состоит из корректных символов
П9	Краткое имя узла высказывания состоит из корректных символов
П10	Краткое имя находится в единственном экземпляре
С1	Сообщение: «Отсутствует начальный узел»
С2	Сообщение: «Отсутствует конечный узел»
С3	Сообщение: «Отсутствует узел высказывания»
С4	Сообщение: «Некорректная длина краткого имени начального узла»
С5	Сообщение: «Некорректная длина краткого имени конечного узла»
С6	Сообщение: «Некорректная длина краткого имени узла высказывания»
С7	Сообщение: «Некорректные символы в строке краткого имени начального узла»
С8	Сообщение: «Некорректные символы в строке краткого имени конечного узла»
С9	Сообщение: «Некорректные символы в строке краткого имени узла высказывания»
С10	Сообщение: «Узел с таким кратким именем уже существует»
С11	Сообщение: «Диалог успешно создан»

В качестве графического представление причин и следствий приведена функциональная диаграмма на рисунке 34.

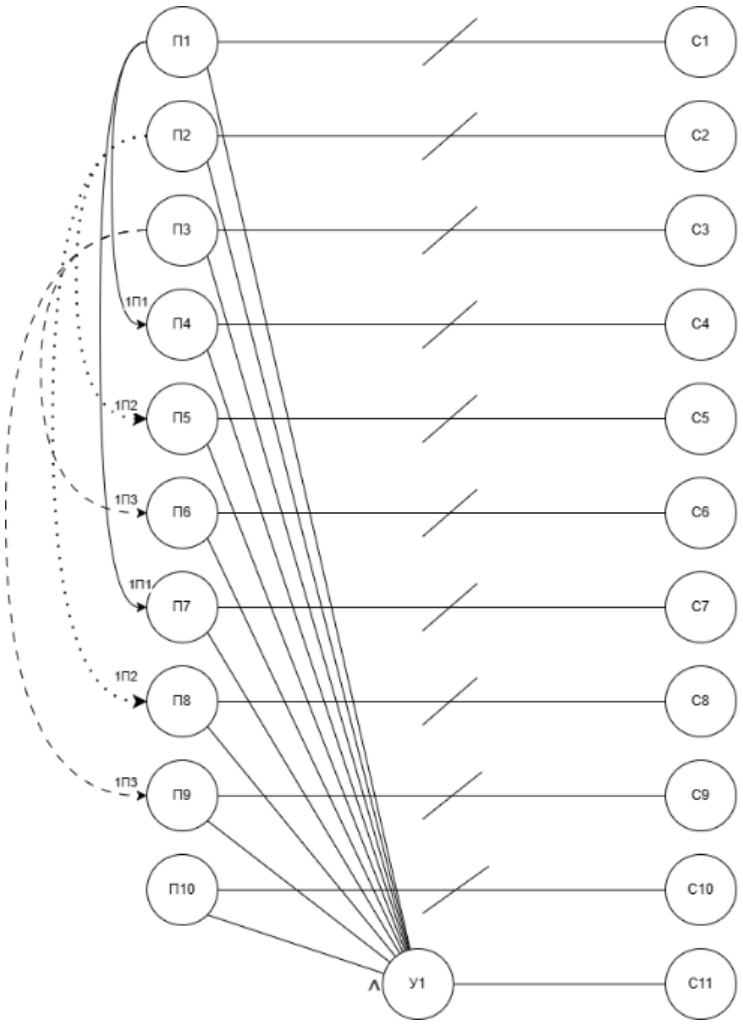

Рисунок 34 – Функциональная диаграмма

Таблицы с представлением допустимых ограничений в случае полного перебора всех комбинаций определяют 1024 тестовых сценария. С целью сокращения количества вхождений во множество допустимых тестов используется комбинаторный метод тестирования программного обеспечения, называемый попарным тестированием, который для каждой пары входных параметров системы проверяет все возможные дискретные комбинации этих параметров. Данная техника позволяет уменьшить количество тестовых сценариев с минимальным вмешательством в качество проводимого тестирования.

Для проработки каждого тестируемого варианта, была применена техника попарного тестирования, которая предоставит все возможные варианты тестирования. Результат применения техники попарного тестирования оставил

из 1024 тестовых сценариев лишь 51. Тестовые сценарии после применения техники попарного тестирования представлены на таблице 7.

Таблица 7 – Тестовые сценарии после применения техники попарного тестирования

	П1	**П2**	**П3**	**П4**	**П5**	**П6**	**П7**	**П8**	**П9**	**П10**
1	Да	Да	Да	2 – 8	9 – ∞	0 – 1	@, №, ! , ()	0..9	@, №, ! , ()	Да
2	Да	Да	Да	0 – 1	2 – 8	9 – ∞	0..9	@, №, ! , ()	0..9	Нет
3	Да	Нет	Нет	9 – ∞	0 – 1	2 – 8	A..Z	A..Я	A..Z	Да
4	Нет	Нет	Да	2 – 8	2 – 8	2 – 8	A..Я	@, №, ! , ()	A..Z	Да
5	Нет	Да	Нет	0 – 1	0 – 1	0 – 1	0..9	A..Я	@, №, ! , ()	Нет
6	Нет	Нет	Да	9 – ∞	9 – ∞	9 – ∞	A..Z	0..9	A..Я	Да
7	Нет	Да	Да	2 – 8	2 – 8	2 – 8	@, №, ! , ()	A..Z	A..Я	Нет
8	Нет	Нет	Нет	0 – 1	0 – 1	0 – 1	A..Я	@, №, ! , ()	0..9	Да
9	Нет	Да	Да	9 – ∞	9 – ∞	9 – ∞	0..9	A..Я	A..Z	Нет
10	Нет	Нет	Да	0 – 1	0 – 1	0 – 1	@, №, ! , ()	0..9	A..Я	Нет
11	Да	Да	Нет	9 – ∞	2 – 8	0 – 1	A..Я	@, №, ! , ()	@, №, ! , ()	Да
12	Да	Да	Да	9 – ∞	2 – 8	0 – 1	A..Z	A..Z	0..9	Нет
13	Да	Нет	Нет	2 – 8	0 – 1	9 – ∞	@, №, ! , ()	@, №, ! , ()	A..Z	Да
14	Да	Да	Да	0 – 1	9 – ∞	2 – 8	A..Я	A..Я	@, №, ! , ()	Нет
15	Да	Нет	Нет	9 – ∞	2 – 8	0 – 1	0..9	A..Я	A..Я	Да
16	Нет	Нет	Нет	2 – 8	9 – ∞	0 – 1	@, №, ! , ()	A..Я	A..Я	Да
17	Нет	Да	Да	0 – 1	2 – 8	9 – ∞	A..Я	A..Я	0..9	Нет
18	Нет	Нет	Нет	9 – ∞	0 – 1	2 – 8	0..9	0..9	A..Z	Да
19	Нет	Да	Да	2 – 8	9 – ∞	0 – 1	A..Z	A..Z	@, №, ! , ()	Нет
20	Нет	Да	Нет	0 – 1	2 – 8	9 – ∞	@, №, ! , ()	@, №, ! , ()	A..Я	Да
21	Нет	Нет	Да	9 – ∞	0 – 1	2 – 8	A..Я	A..Я	A..Я	Нет
22	Нет	Да	Нет	2 – 8	9 – ∞	0 – 1	A..Я	0..9	0..9	Да
23	Нет	Нет	Да	0 – 1	2 – 8	9 – ∞	0..9	A..Z	A..Z	Нет
24	Нет	Да	Да	9 – ∞	0 – 1	2 – 8	A..Z	@, №, ! , ()	@, №, ! , ()	Да
25	Да	Да	Нет	2 – 8	2 – 8	2 – 8	A..Я	A..Z	@, №, ! , ()	Да
26	Да	Нет	Да	0 – 1	0 – 1	0 – 1	0..9	@, №, ! , ()	A..Я	Нет
27	Да	Да	Нет	9 – ∞	9 – ∞	9 – ∞	A..Z	A..Я	0..9	Да
28	Да	Да	Да	2 – 8	2 – 8	2 – 8	@, №, ! , ()	A..Я	A..Z	Нет
29	Да	Нет	Нет	0 – 1	0 – 1	0 – 1	A..Я	0..9	@, №, ! , ()	Да

Продолжение таблицы 7

	П1	П2	П3	П4	П5	П6	П7	П8	П9	П10
30	Да	Да	Да	9 – ∞	9 – ∞	9 – ∞	0..9	A..Z	A..Я	Нет
31	Да	Нет	Да	2 – 8	2 – 8	2 – 8	A..Z	@, №, ! , ()	A..Я	Да
32	Да	Да	Нет	0 – 1	0 – 1	0 – 1	@, №, ! , ()	A..Я	0..9	Нет
33	Да	Нет	Да	9 – ∞	9 – ∞	9 – ∞	A..Я	0..9	A..Z	Да
34	Нет	Нет	Нет	2 – 8	0 – 1	9 – ∞	A..Z	A..Я	A..Z	Да
35	Нет	Да	Да	0 – 1	9 – ∞	2 – 8	@, №, ! , ()	0..9	@, №, ! , ()	Нет
36	Нет	Да	Нет	9 – ∞	2 – 8	0 – 1	A..Я	A..Z	A..Я	Да
37	Нет	Нет	Да	2 – 8	0 – 1	9 – ∞	A..Я	@, №, ! , ()	0..9	Нет
38	Нет	Да	Да	0 – 1	9 – ∞	2 – 8	0..9	A..Я	A..Z	Да
39	Нет	Нет	Нет	9 – ∞	2 – 8	0 – 1	A..Z	A..Я	@, №, ! , ()	Нет
40	Нет	Да	Да	2 – 8	0 – 1	9 – ∞	@, №, ! , ()	0..9	A..Я	Да
41	Нет	Нет	Нет	0 – 1	9 – ∞	2 – 8	A..Я	A..Z	A..Я	Нет
42	Нет	Да	Да	9 – ∞	2 – 8	0 – 1	0..9	@, №, ! , ()	0..9	Да
43	Да	Да	Нет	2 – 8	9 – ∞	0 – 1	@, №, ! , ()	A..Z	0..9	Да
44	Да	Да	Да	0 – 1	2 – 8	9 – ∞	A..Я	@, №, ! , ()	A..Z	Нет
45	Да	Нет	Да	9 – ∞	0 – 1	2 – 8	0..9	A..Я	@, №, ! , ()	Да
46	Да	Да	Нет	2 – 8	9 – ∞	0 – 1	A..Z	0..9	A..Я	Нет
47	Да	Нет	Да	0 – 1	2 – 8	9 – ∞	@, №, ! , ()	A..Z	0..9	Да
48	Да	Да	Нет	9 – ∞	0 – 1	2 – 8	A..Я	@, №, ! , ()	A..Z	Нет
49	Да	Нет	Да	2 – 8	9 – ∞	0 – 1	A..Я	A..Я	@, №, ! , ()	Да
50	Да	Да	Нет	0 – 1	2 – 8	9 – ∞	0..9	A..Я	A..Я	Нет
51	Да	Нет	Да	9 – ∞	0 – 1	2 – 8	A..Z	0..9	A..Я	Да

Результат выполнения каждого тестового сценария, определенного на таблице 8, привел к следствию с сообщением «Диалог успешно создан».

Таким образом, можно сделать вывод о том, что результат выполнения технического задания произведен успешно и разработанное программное обеспечение работает корректно.

4.2 Анализ сложности кода модуля создания узлов

Для измерения характеристик и критериев качества используют метрики. Метрика качества программ – это система измерений качества программ. Эти измерения могут проводиться на уровне критериев качества программ или на уровне отдельных характеристик качества. В первом случае система измерений позволяет непосредственно сравнивать программы по качеству, при этом сами измерения не могут быть проведены без субъективных оценок свойств программ. Во втором случае измерения характеристик можно выполнить объективно и достоверно, но оценка качества ПО в целом будет связана с субъективной интерпретацией получаемых оценок. В листинге 6 приведён код, проверяющий корректность введённого email или пароля, на основании которого будет производится расчёт метрик.

Листинг 6 – Функция размещения узла

```
const onDrop = (event) => {
  // Предотвращение стандартного поведения браузера при
перетаскивании
  event.preventDefault();

  // Получение границ контейнера ReactFlow для расчета позиции
  const reactFlowBounds =
reactFlowWrapper.current.getBoundingClientRect();

  // Получение типа узла из данных перетаскивания
  const type =
event.dataTransfer.getData('application/reactflow');

  // Преобразование координат мыши в координаты графа
  const position = reactFlowInstance.project({
    x: event.clientX  -  reactFlowBounds.left,
    y: event.clientY  -  reactFlowBounds.top,
  });

  // Создание нового узла в зависимости от типа
  const newNode =
    type === 'start'
      ? new StartNode(getId(), position, 'Вход')
      : type === 'end'
        ? new EndNode(getId(), position, 'Выход')
        : new StatementNode(getId(), position, 'Текст', 'Текст',
'Полный текст');

  // Добавление нового узла в список узлов
```

```
  setNodes((nds) => nds.concat(newNode));

  // Если создан узел высказывания, инициализируем его данные в
хранилище
  if (newNode instanceof StatementNode) {
    statementDataStorage[newNode.id] = {
      shortName: newNode.data.shortName,
      fullText: newNode.data.fullText,
    };
  }
};
```

Расчет метрики SLOC на примере кода функции из листинга 1 представлен на таблице 8.

Таблица 8 – Расчет метрики SLOC для кода из листинга 1

Метрика	**Результат**
Физические строки кода	35
Исходные строки кода	22
Строки комментариев	7
Пустые строки	6

Результат расчета метрики показывает, что код функции, представленный на листинге n, не имеет достаточной сложности, его просто анализировать и использовать.

Приведённый фрагмент кода состоит из предотвращения стандартного поведения браузера при перетаскивании, получения границ контейнера и типа создаваемого узла. Затем координаты мыши преобразуются в координаты холста для корректного размещения. После этого, в зависимости от выбранного типа, мы создаём новый узел, объявляя экземпляр соответствующего класса. Созданный узел добавляется в список узлов и, если это узел высказывания, то инициализируем краткое имя и полный текст в хранилище.

Традиционной характеристикой размера программ является количество строк исходного текста. Под строкой понимается любой оператор программы,

поскольку именно оператор, а не отдельно взятая строка является тем интеллектуальным «квантом» программы, опираясь на который можно строить метрики сложности ее создания.

Оценка размера программы недостаточна для принятия решения о ее сложности, но вполне применима для классификации программ, существенно различающихся объемами. При уменьшении различий в объеме программ на первый план выдвигаются оценки других факторов, оказывающих влияние на сложность. Таким образом, оценка размера программы есть оценка по номинальной шкале, на основе которой определяются только категории программ без уточнения оценки для каждой категории. К группе оценок размера программ можно отнести метрику Холстеда.

Основу метрики Холстеда составляют четыре измеряемых характеристики программы:

n1 – число уникальных операторов программы, включая символы разделители, имена процедур и знаки операций (словарь операторов), n2 – число уникальных операндов программы (словарь операндов)

N1 – общее число операторов в программе, N2 – общее число операндов в программе.

В таблице 9 приведены результаты расчёта количества операторов и операндов для тестируемого блока кода.

Таблица 9 – Операторы и операнды

№ п/п	Операторы	Число операторов	Операнды	Число операндов
1	=	5	event	4
2	const	5	reactFlowBounds	3
3	=>	2	reactFlowWrapper	1
4	‘	16	current	1
5	{	4	type	3
6	(	5	dataTransfer	1
7	.	16	'application/reactflow'	1
8	}	4	position	4
9	;	8	reactFlowInstance	1
10	,	13	statementDataStorage	1
11	if	1	data	2
12	=>	2	clientX	1
13	?	2	clientY	1
14	:	6	left	1
15	concat	1	top	1

Продолжение таблицы 9

№ п/п	Операторы	Число операторов	Операнды	Число операндов
16	===	2	newNode	6
17	[	1	shortName	2
18	instanceof	1	fullText	2
19	)	5	id	1
20	]	1	StatementNode	2
21	new	3	EndNode	2
22	getBoundingClientRect	1	StartNode	2
23	getData	1	'Текст'	2
24	preventDefault	1	'Полный текст'	1
25	project	1	'Вход'	1
26			'Выход'	1
27			'start'	1
28			'end'	1
29			nds	2

Произведем расчет метрики Холстеда на примере кода из листинга 6.

Длина кода программы рассчитывается по формуле:

$$N=N1+N2=107+52=159,$$

где N – длина программы;

N_1 – общее число операторов в программе;

N_2 – общее число операндов в программе.

Словарь программы рассчитывается по формуле:

$$\eta=\eta_1+\eta_2=25+29=54,$$

где η – словарь программы;

η_1 – число уникальных операторов программы;

η_2 – число уникальных операндов программы.

Потенциальный объем программы рассчитывается по формуле:

$$\widehat{N}=\eta_1 \log_2 \eta_1 +\eta_2 \log_2 \eta_2=256.97,$$

где η_1 – число уникальных операторов программы;

η_2 – число уникальных операндов программы.

Вычислительная сложность программы по метрике Холстеда вычисляется по формуле:

$$V=N \log_2 \eta=915.02,$$

где V – вычислительная сложность программы по метрике Холстеда;

N – длина программы;

η – словарь программы.

Сложность кода программы рассчитывается по формуле:

$$D = \frac{\eta_1}{2} * \frac{N_2}{\eta_2} = 22.41,$$

где D – сложность кода программы;

N_2 – общее число операндов в программе;

η_1 – число уникальных операторов программы;

η_2 – число уникальных операндов программы.

Усилие для написания и понимания кода рассчитывается по формуле:

$$E=D*V=20505.59,$$

где E – Усилие для написания и понимания кода;

D – сложность кода программы;

V – вычислительная сложность программы по метрике Холстеда.

Время написания кода рассчитывается по формуле:

$$T = \frac{E}{18} = 1139.19,$$

где T – Время написания кода рассчитывается по формуле:

E – Усилие для написания и понимания кода.

Количество доставленных кодом ошибок рассчитывается по формуле:

$$B = \frac{V}{3000} = 0.3,$$

где B – Количество доставленных кодом ошибок;

V – вычислительная сложность программы по метрике Холстеда.

В результате расчёта сложности кода по метрике Холстеда вышло, что программный код рассматриваемой функции удовлетворяет требуемым ограничениям на значения метрик качества. Удовлетворение требуемым ограничениям определяет код рассматриваемой программы как успешный вариант реализации.

4.3 Модульное тестирование

Для проверки корректности работы отдельных модулей и компонентов

разработанного веб-приложения было проведено модульное тестирование. Этот подход позволил в автоматическом режиме проверить корректность работы React-компонентов, их взаимодействия, а также утилитных функций. Кроме того, это обеспечило возможность экономии времени при дальнейшем развитии проекта и рефакторинге.

Для реализации модульных тестов компонентов пользовательского интерфейса приложения был выбран фреймворк тестирования Jest – популярный JavaScript-фреймворк, широко используемый для тестирования React-приложений [20].

В процессе написания тестов активно применялись Moq (мок-объекты), которые позволяют имитировать функциональность тестируемого модуля. Это необходимо для изоляции тестируемого кода, чтобы убедиться, что тест проверяет только логику самого модуля, а не поведение его зависимостей. Для проверки ожидаемого поведения кода и результатов его выполнения в Jest использовались мэтчеры (matchers).

Приложение В содержит пример исходного кода модульных тестов, для основных компонентов приложения, результаты проверки которых представлены на рисунке 35.

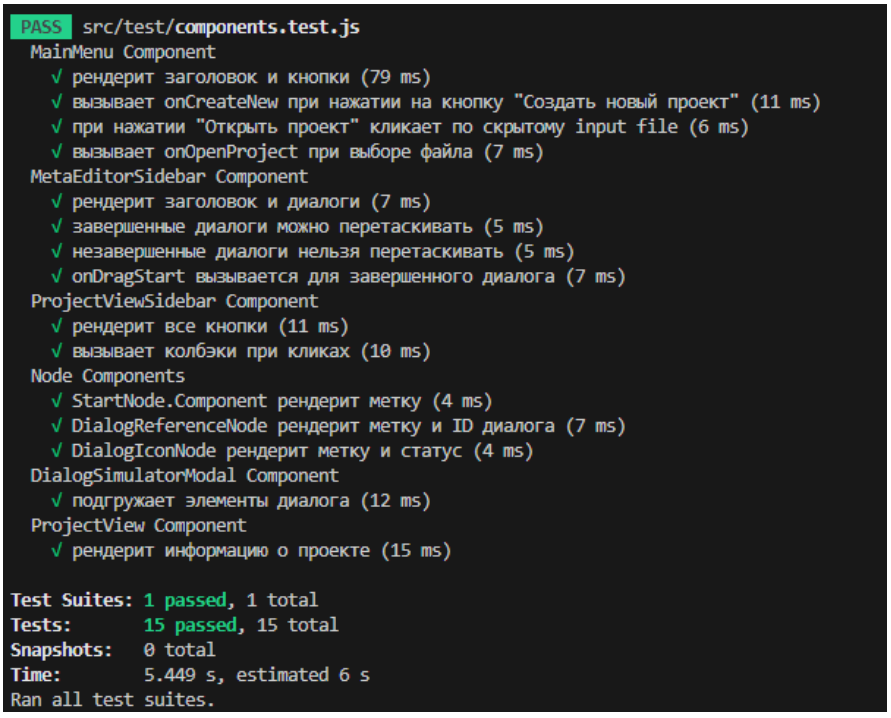

Рисунок 35 – Результаты прохождения модульных тестов

Успешное прохождение тестов, свидетельствует о надежности проверенных модулей и закладывает основу для эффективного сопровождения и дальнейшего развития проекта.

Таким образом, в рамках данного раздела было проведено всестороннее тестирование веб-инструмента по созданию и редактированию узлов для игровых приложений. Методом чёрного ящика была подтверждена корректность функциональности создания диалогов и обработки пользовательского ввода. Анализ сложности кода показал, что код модуля является качественным, не обладает избыточной сложностью и легко поддерживаемым. Тестирование методом белого ящика подтвердило корректность внутренней логики алгоритмов создания узлов. Наконец, модульное тестирование обеспечило проверку работоспособности отдельных React-компонентов и утилитных функций.

Заключение

В ходе выполнения выпускной квалификационной работы был разработан веб-инструмент для создания и редактирования интерактивных диалогов в игровых приложениях. Целью работы являлась разработка программного средства для визуального конструирования и управления графами диалоговых сценариев, обеспечивающего наглядное представление их структуры и удобные инструменты для модификации, которая была успешно достигнута.

Проведен анализ предметной области, изучены материалы по тематике разработки. Осуществлен сравнительный анализ существующих программных продуктов-аналогов, выявлены их сильные и слабые стороны. На основании полученных результатов были сформулированы и проанализированы функциональные и нефункциональные требования к разрабатываемому веб-инструменту.

Выполнено проектирование программных средств. Разработана трехуровневая архитектура приложения. Спроектирована структура хранения данных для диалоговых сценариев в формате JSON. Спроектирован пользовательский интерфейс веб-инструмента.

Осуществлена реализация программного продукта. Разработка велась на языке программирования JavaScript с использованием библиотеки React для построения пользовательского интерфейса и специализированной библиотеки React Flow для визуализации и управления графовыми структурами диалогов. Реализованы основные функции, включая создание и редактирование интерактивных диалогов в виде графа, экспорт проектов в формат JSON и импорт из него, управление проектами диалогов, симуляцию диалогов и гибкую настройку интерфейса.

Проведено тестирование разработанного программного обеспечения. Выполнено функциональное тестирование, расчет и оценка метрик качества кода и осуществлено автоматизированное тестирование производительности ключевых операций.

Выполнено планирование разработки и оценка бюджета проекта с

использованием Microsoft Project. Установлено, что длительность проекта составила 34 рабочих дня, а суммарные затраты – 146 952,00 рублей.

Таким образом, в результате выполнения выпускной квалификационной работы разработан веб-инструмент для создания и редактирования интерактивных диалогов в игровых приложениях. Результаты данной работы могут быть полезны разработчикам игр, сценаристам и геймдизайнерам для упрощения и повышения эффективности процесса создания сложных и разветвленных диалоговых систем.

Список использованных источников

1. Twine справочник // Twinery [Электронный ресурс]. URL: https://twinery.org/reference/en/ (дата обращения: 09.04.2025).

2. Articy:draft : страница сообщества // Сообщество Steam [Электронный ресурс]. URL: https://steamcommunity.com/app/230780 (дата обращения: 09.04.2025).

3. draw.io Documentation // Draw.io [Электронный ресурс]. URL: https://www.drawio.com/doc/ (дата обращения: 09.04.2025).

4. Documentation // ChatMapper [Электронный ресурс]. URL: https://www.chatmapper.com/documentation/ (дата обращения: 10.04.2025).

5. YarnEditor // GitHub [Электронный ресурс]. URL: https://github.com/gamemagics/YarnEditor/ (дата обращения: 11.04.2025).

6. Диаграмма прецедентов // System Analyst Base [Электронный ресурс]. URL: https://docs.system-analyst-base.ru/hard-skills/proektirovanie/notacii-i-diagrammy/uml/diagramma-precedentov-use-case/ (дата обращения: 15.04.2025).

7. Single Page Application: как работает сайт-приложение // Hexlet [Электронный ресурс]. URL: https://ru.hexlet.io/blog/posts/single-page-application-kak-rabotaet-sayt-prilozhenie/ (дата обращения: 17.04.2025). (Примечание: удален параметр ysclid из URL)

8. Tutorial : React Router v6.30.1 // React Router [Электронный ресурс]. URL: https://reactrouter.com/6.30.1/start/tutorial/ (дата обращения: 18.04.2025).

9. Начало работы // React [Электронный ресурс]. URL: https://ru.react.js.org/docs/getting-started.html/ (дата обращения: 23.04.2025).

10. Quickstart // React Flow [Электронный ресурс]. URL: https://reactflow.dev/learn/ (дата обращения: 28.04.2025).

11. Простое руководство по диаграмме компонентов // Creately [Электронный ресурс]. URL: https://creately.com/blog/ru/uncategorized-ru/учебное-пособие-по-компонентной-диаг/ (дата обращения: 01.05.2025).

12. Фаулер, М. Архитектура корпоративных программных приложений / М. Фаулер ; пер. с англ. – М. : Издательский дом «Вильямс», 2006. – 544 с.

13. Introducing JSON // json.org [Электронный ресурс]. URL: https://www.json.org/json-en.html/ (дата обращения: 04.05.2025).

14. Диаграмма классов // ENSI.tech [Электронный ресурс]. URL: https://docs.ensi.tech/analyst-guides/tools/diagrams/uml/class-diagram/ (дата обращения: 06.05.2025).

15. Проектирование диаграммы деятельности UML (Activity Diagram) // WorldSkills Russia National Team [Электронный ресурс]. URL: https://nationalteam.worldskills.ru/skills/proektirovanie-diagrammy-deyatelnosti-uml-activity-diagram/ (дата обращения: 10.05.2025).

16. Диаграмма последовательности (sequence-диаграмма) // Хабр [Электронный ресурс]. URL: https://habr.com/ru/articles/814769/ (дата обращения: 15.05.2025).

17. Get started // Figma Help Center [Электронный ресурс]. URL: https://help.figma.com/hc/en-us/categories/360002051613-Get-started/ (дата обращения: 18.05.2025).

18. Что такое диаграммы Ганта? // Atlassian [Электронный ресурс]. URL: https://www.atlassian.com/ru/agile/project-management/gantt-chart/ (дата обращения: 20.05.2025).

19. Майерс, Г. Искусство тестирования программ / Г. Майерс, Т. Баджетт, К. Сандлер ; пер. с англ. – 3-е изд. – М. : ООО «И.Д. Вильямс», 2012. – 272 с.

20. Getting Started // Jest [Электронный ресурс]. URL: https://jestjs.io/docs/getting-started/ (дата обращения: 25.05.2025).

Приложение А
(справочное)
Глоссарий

Jest – популярный JavaScript-фреймворк для тестирования, широко используемый для тестирования React-приложений. Применялся для написания и проведения модульных тестов.

JSON (JavaScript Object Notation) – текстовый формат обмена данными, основанный на синтаксисе JavaScript. В работе используется как основной формат для экспорта и импорта структуры диалоговых графов и проектов.

Moq (Mock Object) – объект-заглушка или имитатор, используемый в модульном тестировании для изоляции тестируемого компонента путем имитации поведения его зависимостей.

React – JavaScript-библиотека для создания пользовательских интерфейсов, основанная на компонентном подходе. Использована для разработки слоя представления веб-инструмента.

React Flow – специализированная библиотека для React, предназначенная для создания интерактивных графовых редакторов и диаграмм. Использована для визуализации и управления графами диалогов.

SPA (Single Page Application) – одностраничное приложение, веб-приложение, которое загружает одну HTML-страницу и динамически обновляет её содержимое по мере взаимодействия с пользователем, вместо загрузки новых страниц с сервера. Разработанный веб-инструмент реализован как SPA.

UML (Unified Modeling Language) – унифицированный язык моделирования, используемый для графического описания и объектного моделирования в разработке программного обеспечения, бизнес-процессов, системного проектирования и отображения организационных структур. В работе применялся для создания диаграмм вариантов использования, классов, деятельности и последовательности.

UUID (Universally Unique Identifier) – универсальный уникальный идентификатор, стандарт, используемый для генерации идентификаторов, которые с высокой вероятностью будут уникальными. В работе используется для присвоения ID узлам, ребрам и другим элементам.

Мэтчер (Matcher) – в контексте фреймворка Jest, функция, используемая для проверки соответствия фактического значения ожидаемому значению в тестах.

Хук (React) – специальные функции в React, позволяющие использовать состояние и другие возможности React в функциональных компонентах без написания классов.

Приложение Б

(обязательное)

Листинг программы

Листинг Б.1 – Файл App.js

```
import React, { useState, useRef, useEffect, useCallback, useMemo,
forwardRef, useImperativeHandle } from 'react';
import ReactFlow, {
  ReactFlowProvider,
  Controls,
  Background,
  useNodesState,
  useEdgesState,
  addEdge,
} from 'react-flow-renderer';
import NodeEditor from './NodeEditor';
import './index.css';
import { v4 as uuidv4 } from 'uuid';
import * as htmlToImage from 'html-to-image';
import DialogSimulatorModal from
'./components/DialogSimulatorModal';
import ValidationErrorsModal from
'./components/ValidationErrorsModal';
import { validateDialogGraph, validateMetaGraph } from
'./utils/validationUtils';
import { deserializeNodes, getDialogEntryPoints,
getDialogExitPoints, serializeNodes,
serializeEdges,deserializeEdges } from './utils/graphUtils';
const LIGHT_BACKGROUND_APP = '#FFFFFF';
const DARK_TEXT_COLOR_APP = '#1A1A1A';
const FAT_BORDER_COLOR_APP = '#1A1A1A';
const BORDER_WIDTH_APP = '2px';
const ACCENT_COLOR_PRIMARY_APP = '#007ACC';
const FONT_FAMILY_TEXT_APP = "'Arial', sans-serif";
const FONT_FAMILY_HEADINGS_APP = "'Arial Black', 'Impact', sans-
serif";
const BUTTON_SHADOW_APP = `2px 2px 0px ${FAT_BORDER_COLOR_APP}`;
const BUTTON_TRANSFORM_INITIAL_APP = 'translate(-1px, -1px)';
const BUTTON_SHADOW_HOVER_APP = `1px 1px 0px
${FAT_BORDER_COLOR_APP}`;
const BUTTON_TRANSFORM_HOVER_APP = 'translate(0px, 0px)';
const BUTTON_SHADOW_ACTIVE_APP = `0px 0px 0px
${FAT_BORDER_COLOR_APP}`;
const BUTTON_TRANSFORM_ACTIVE_APP = 'translate(1px, 1px)';
const App = forwardRef(({
  initialNameProp,
  initialNodesProp,
  initialEdgesProp,
  initialStatementDataProp,
  initialOptionDataProp,
  sidebarToRender,
  nodeTypesToUse,
  onSaveCallback,
  appEditingContext,
  onAppSaveChangesAndBack,
  onOpenDialogFromMeta,
```

```
  projectDataForSimulator,
}, ref) => {
  const reactFlowWrapper = useRef(null);
  const [reactFlowInstance, setReactFlowInstance] = useState(null);
  const [nodes, setNodes, onNodesChange] =
useNodesState(initialNodesProp || []);
  const [edges, setEdges, onEdgesChange] =
useEdgesState(initialEdgesProp || []);
  const [currentName, setCurrentName] = useState(initialNameProp ||
"Редактор");
  const [statementData, setStatementData] =
useState(initialStatementDataProp || {});
  const [optionData, setOptionData] =
useState(initialOptionDataProp || {});
  const [selectedNode, setSelectedNode] = useState(null);
  const [selectedOptionId, setSelectedOptionId] = useState(null);
  const [isConnecting, setIsConnecting] = useState(false);
  const [showSaveDialogioMenu, setShowSaveDialogioMenu] =
useState(false);
  const [showExportMenu, setShowExportMenu] = useState(false);
  const [isSimulatorOpen, setIsSimulatorOpen] = useState(false);
  const [validationIssues, setValidationIssues] = useState([]);
  const [isValidationModalOpen, setIsValidationModalOpen] =
useState(false);
  const [simulatorInitialProps, setSimulatorInitialProps] =
useState({ nodeId: null, entryPoints: null });
  useImperativeHandle(ref, () => ({
    getGraphData: () => ({
      name: currentName, nodes, edges,
      statementDataStorage: statementData,
      optionDataStorage: optionData,
    })
  }));
  const resetAllErrorHighlighting = useCallback(() => {
    setNodes(nds => nds.map(n => ({
      ...n,
      data: { ...n.data, isError: false, isWarning: false,
errorOptionIds: undefined },
      style: { ...(n.style || {}), border: n.data?.originalBorder
|| (n.type === 'dialogRef' ? `${BORDER_WIDTH_APP} solid #8a6d3b` :
`${BORDER_WIDTH_APP} solid ${FAT_BORDER_COLOR_APP}`) }
    })));
    setEdges(eds => eds.map(e => ({
      ...e,
      style: {
        ...(e.style || {}),
        stroke: e.data?.originalStroke || FAT_BORDER_COLOR_APP,
        strokeWidth: e.data?.originalStrokeWidth || 2,
      }
    })));
    setValidationIssues([]);
  }, [setNodes, setEdges]);
  useEffect(() => {
```

```
    setNodes(initialNodesProp || []);
    setEdges(initialEdgesProp || []);
    setCurrentName(initialNameProp || "Редактор");
    setStatementData(initialStatementDataProp || {});
    setOptionData(initialOptionDataProp || {});
    setSelectedNode(null);
    setSelectedOptionId(null);
    setShowSaveDialogioMenu(false);
    setShowExportMenu(false);
    resetAllErrorHighlighting();
  }, [initialNameProp, initialNodesProp, initialEdgesProp,
initialStatementDataProp, initialOptionDataProp,
resetAllErrorHighlighting]);
  const memoizedNodeTypes = useMemo(() => {
    if (!nodeTypesToUse) return {};
    const baseTypes = {
      ...(nodeTypesToUse.start && { start: (props) =>
<nodeTypesToUse.start {...props} selected={selectedNode?.id ===
props.id} data={props.data} /> }),
      ...(nodeTypesToUse.end && { end: (props) =>
<nodeTypesToUse.end {...props} selected={selectedNode?.id ===
props.id} isConnecting={isConnecting} data={props.data} /> }),
      ...(nodeTypesToUse.statement && { statement: (props) =>
<nodeTypesToUse.statement {...props} selected={selectedNode?.id ===
props.id} isConnecting={isConnecting}
setSelectedOptionId={setSelectedOptionId}
selectedOptionId={selectedOptionId} data={props.data} /> }),
      ...(nodeTypesToUse.dialogRef && {
        dialogRef: (props) => (
          <nodeTypesToUse.dialogRef
            {...props}
            selected={selectedNode?.id === props.id}
            data={props.data}
          />
        )
      }),
    };
    return baseTypes;
  }, [nodeTypesToUse, selectedNode?.id, isConnecting,
selectedOptionId]);
  useEffect(() => {
    if (selectedNode) {
      const updatedNode = nodes.find(n => n.id ===
selectedNode.id);
      if (updatedNode && (!selectedNode ||
JSON.stringify(updatedNode) !== JSON.stringify(selectedNode))) {
        setSelectedNode(updatedNode);
      } else if (!updatedNode && selectedNode) {
        setSelectedNode(null);
        setSelectedOptionId(null);
      }
    }
  }, [nodes, selectedNode]);
```

```
  const handleSaveToProject = useCallback(() => {
    if (onSaveCallback) {
      onSaveCallback({ name: currentName, nodes, edges,
statementDataStorage: statementData, optionDataStorage: optionData
});
      alert("Изменения сохранены в данных проекта.");
    }
  }, [currentName, nodes, edges, statementData, optionData,
onSaveCallback]);
  const handleBackToProjectView = useCallback(() => {
    if (onAppSaveChangesAndBack) onAppSaveChangesAndBack();
  }, [onAppSaveChangesAndBack]);
  const promptForFileName = (defaultName, extension) => {
    const saneDefaultName = (defaultName ||
'graph').replace(/[<>:"/\\|?*]+/g, '_').replace(/\s+/g,
'_').toLowerCase();
    const name = prompt(`Введите имя файла для сохранения (без
расширения .${extension}):`, saneDefaultName);
    return name ? `${name}.${extension}` : null;
  };
  const exportGraphToDialogio = useCallback(() => {
    const fileName = promptForFileName(currentName, 'dialogio');
    if (!fileName) return;
    const graphData = { name: currentName, nodes:
serializeNodes(nodes), edges: serializeEdges(edges),
statementDataStorage: statementData, optionDataStorage: optionData
};
    const dataString = JSON.stringify(graphData, null, 2);
    const xmlString =
`<dialogio><![CDATA[\n${dataString}\n]]></dialogio>`;
    const blob = new Blob([xmlString], { type: 'application/xml'
});
    const url = URL.createObjectURL(blob);
    const link = document.createElement('a');
    link.href = url;
    link.download = fileName;
    document.body.appendChild(link);
    link.click();
    document.body.removeChild(link);
    URL.revokeObjectURL(url);
    setShowSaveDialogioMenu(false);
  }, [currentName, nodes, edges, statementData, optionData]);
  const importGraphFromDialogio = useCallback((event) => {
    const file = event.target.files[0];
    if (!file) return;
    const reader = new FileReader();
    reader.onload = (e) => {
      try {
        const xmlString = e.target.result;
        const parser = new DOMParser();
        const xmlDoc = parser.parseFromString(xmlString,
"application/xml");
        const dialogioElement = xmlDoc.querySelector('dialogio');
```

```
        if (!dialogioElement || !dialogioElement.textContent) throw
new Error("Некорректный формат .dialogio.");
        const graphData = JSON.parse(dialogioElement.textContent);
        setCurrentName(graphData.name ||
file.name.replace(/\.dialogio$/i, ''));
        setNodes(deserializeNodes(graphData.nodes || []));
        setEdges(deserializeEdges(graphData.edges || []));
        setStatementData(graphData.statementDataStorage || {});
        setOptionData(graphData.optionDataStorage || {});
        setSelectedNode(null);
        resetAllErrorHighlighting();
        alert("Данные из .dialogio импортированы. Не забудьте
сохранить их в проект, если это необходимо.");
      } catch (err) { alert(`Ошибка импорта .dialogio:
${err.message}`); console.error("Dialogio Import error:", err); }
    };
    reader.readAsText(file);
    event.target.value = null;
    setShowSaveDialogioMenu(false);
  }, [setCurrentName, setNodes, setEdges, setStatementData,
setOptionData, resetAllErrorHighlighting]);
  const exportGraphToJson = useCallback(() => {
    const fileName = promptForFileName(currentName, 'json');
    if (!fileName) return;
    const graphData = { name: currentName, nodes:
serializeNodes(nodes), edges: serializeEdges(edges),
statementDataStorage: statementData, optionDataStorage: optionData
};
    const jsonString = JSON.stringify(graphData, null, 2);
    const blob = new Blob([jsonString], { type: 'application/json'
});
    const url = URL.createObjectURL(blob);
    const link = document.createElement('a');
    link.href = url;
    link.download = fileName;
    document.body.appendChild(link);
    link.click();
    document.body.removeChild(link);
    URL.revokeObjectURL(url);
    setShowExportMenu(false);
  }, [currentName, nodes, edges, statementData, optionData]);
  const importGraphFromJson = useCallback((event) => {
    const file = event.target.files[0];
    if (!file) return;
    const reader = new FileReader();
    reader.onload = (e) => {
      try {
        const importedData = JSON.parse(e.target.result);
        setCurrentName(importedData.name ||
file.name.replace(/\.json$/i, ''));
        setNodes(deserializeNodes(importedData.nodes || []));
        setEdges(deserializeEdges(importedData.edges || []));
        setStatementData(importedData.statementDataStorage || {});
```

```
        setOptionData(importedData.optionDataStorage || {});
        setSelectedNode(null);
        resetAllErrorHighlighting();
        alert("Данные из .json импортированы. Не забудьте сохранить
их в проект, если это необходимо.");
      } catch (err) { alert(`Ошибка импорта JSON: ${err.message}`);
console.error("JSON Import error:", err); }
    };
    reader.readAsText(file);
    event.target.value = null;
    setShowExportMenu(false);
  }, [setCurrentName, setNodes, setEdges, setStatementData,
setOptionData, resetAllErrorHighlighting]);
  const downloadImageWithPrompt = (dataUrl, extension, defaultName)
=> {
    const fileName = promptForFileName(defaultName, extension);
    if (!fileName) return;
    const link = document.createElement('a');
    link.download = fileName;
    link.href = dataUrl;
    document.body.appendChild(link);
    link.click();
    document.body.removeChild(link);
  };
  const exportToImage = useCallback(async (format = 'png') => {
    if (!reactFlowWrapper.current || !reactFlowInstance) {
      console.error("React Flow wrapper или instance не найдены для
экспорта изображения.");
      setShowExportMenu(false);
      return;
    }
    const controls =
reactFlowWrapper.current.querySelector('.react-flow__controls');
    let originalControlsDisplay;
    if (controls) {
      originalControlsDisplay = controls.style.display;
      controls.style.display = 'none';
    }
    const originalEdges = reactFlowInstance.getEdges();
    const edgesWithHigherZIndex = originalEdges.map(edge => ({
      ...edge,
      style: { ...edge.style, zIndex: (edge.style?.zIndex || 1) +
1000 }
    }));
    reactFlowInstance.setEdges(edgesWithHigherZIndex);
    await new Promise(resolve => setTimeout(resolve, 200));
    try {
      let dataUrl;
      const options = { backgroundColor: '#EAEAEA', pixelRatio:
window.devicePixelRatio || 2 };
      if (format === 'png') {
        dataUrl = await htmlToImage.toPng(reactFlowWrapper.current,
options);
```

```
      } else {
        dataUrl = await
htmlToImage.toJpeg(reactFlowWrapper.current, { ...options, quality:
0.95 });
      }
      downloadImageWithPrompt(dataUrl, format, currentName);
    } catch (err) {
      console.error(`Ошибка экспорта в ${format.toUpperCase()}:`,
err);
      alert(`Не удалось экспортировать изображение. Подробности в
консоли.`);
    } finally {
      if (controls) controls.style.display =
originalControlsDisplay;
      reactFlowInstance.setEdges(originalEdges);
      setShowExportMenu(false);
    }
  }, [currentName, reactFlowInstance]);
  const clearErrorForElement = useCallback((elementId, elementType
= 'node') => {
    setNodes(nds =>
      nds.map(n => {
        if (elementType === 'node' && n.id === elementId) {
          return { ...n, data: { ...n.data, isError: false,
isWarning: false } };
        }
        if (elementType === 'option' && n.type === 'statement' &&
n.data.errorOptionIds?.includes(elementId)) {
          const newErrorOptionIds = n.data.errorOptionIds.filter(id
=> id !== elementId);
          return { ...n, data: { ...n.data, errorOptionIds:
newErrorOptionIds } };
        }
        return n;
      })
    );
    if (elementType === 'edge') {
      setEdges(eds =>
        eds.map(e =>
          e.id === elementId
            ? { ...e, style: { ...(e.style || {}), stroke:
e.data?.originalStroke || FAT_BORDER_COLOR_APP, strokeWidth:
e.data?.originalStrokeWidth || 2 } }
            : e
        )
      );
    }
  }, [setNodes, setEdges]);
  const deleteNode = useCallback((nodeIdToDelete) => {
    const nodeToDelete = nodes.find(n => n.id === nodeIdToDelete);
    setNodes(nds => nds.filter(n => n.id !== nodeIdToDelete));
    setEdges(eds => eds.filter(edge => edge.source !==
nodeIdToDelete && edge.target !== nodeIdToDelete));
```

```
    setNodes(nds => nds.map(n => {
      if (n.type === 'statement' && n.data.options) {
        let optionsChanged = false;
        const newOptions = n.data.options.map(opt => {
          if (opt.target === nodeIdToDelete) {
            optionsChanged = true;
            return { ...opt, target: '' };
          }
          return opt;
        });
        if (optionsChanged) {
          const newErrorOptionIds = (n.data.errorOptionIds ||
[]).filter(id => !newOptions.find(opt => opt.id === id &&
opt.target === ''));
          return { ...n, data: { ...n.data, options: newOptions,
errorOptionIds: newErrorOptionIds, isError: n.data.isError &&
newErrorOptionIds.length > 0 } };
        }
      }
      return n;
    }));
    if (nodeToDelete?.type === 'statement' &&
nodeToDelete.data.options) {
      setOptionData(prevOptData => {
        const newOptData = { ...prevOptData };
        nodeToDelete.data.options.forEach(opt => delete
newOptData[opt.id]);
        return newOptData;
      });
    }
    setStatementData(prevStmtData => {
      const newStmtData = { ...prevStmtData };
      delete newStmtData[nodeIdToDelete];
      return newStmtData;
    });
    if (selectedNode?.id === nodeIdToDelete) {
      setSelectedNode(null);
      setSelectedOptionId(null);
    }
    setValidationIssues(prev => prev.filter(issue => issue.id !==
nodeIdToDelete && issue.contextNodeId !== nodeIdToDelete));
  }, [nodes, selectedNode, setNodes, setEdges, setOptionData,
setStatementData, setValidationIssues]);
  const deleteOption = useCallback((nodeId, optionId) => {
    setNodes(nds => nds.map(n => {
      if (n.id === nodeId && n.data.options) {
        const newErrorOptionIds = (n.data.errorOptionIds ||
[]).filter(id => id !== optionId);
        return {
          ...n,
          data: {
            ...n.data,
```

```
            options: n.data.options.filter(opt => opt.id !==
optionId),
            errorOptionIds: newErrorOptionIds,
            isError: n.data.isError && newErrorOptionIds.length > 0
          }
        };
      }
      return n;
    }));
    setEdges(eds => eds.filter(e => !(e.source === nodeId &&
e.sourceHandle === `bottom-${optionId}`)));
    setOptionData(prev => { const newData = { ...prev }; delete
newData[optionId]; return newData; });
    if (selectedOptionId === optionId) setSelectedOptionId(null);
    setValidationIssues(prev => prev.filter(issue => !(issue.type
=== 'option' && issue.id === optionId)));
  }, [selectedOptionId, setNodes, setEdges, setOptionData,
setValidationIssues]);
  const onConnect = useCallback((params) => {
    const sourceHandle = params.sourceHandle || `source-default-
${params.source}`;
    const targetHandle = params.targetHandle || `target-default-
${params.target}`;
    const newEdge = { id: `edge-${uuidv4()}`, ...params,
sourceHandle, targetHandle, type: 'smoothstep', style: { zIndex:
100, strokeWidth: 2, stroke: FAT_BORDER_COLOR_APP } };
    setEdges((eds) => addEdge(newEdge, eds));
    setNodes(nds => nds.map(n => {
      let nodeChanged = false;
      let newData = { ...n.data };
      if (n.id === params.source) {
        nodeChanged = true;
        if (n.type === 'start' || (n.type === 'statement' &&
!sourceHandle.startsWith('bottom-'))) {
          newData.target = params.target;
        } else if (n.type === 'statement' &&
sourceHandle.startsWith('bottom-')) {
          const optionId = sourceHandle.replace('bottom-', '');
          newData.options = n.data.options.map(opt => opt.id ===
optionId ? { ...opt, target: params.target } : opt);
          if (newData.errorOptionIds?.includes(optionId)) {
            newData.errorOptionIds =
newData.errorOptionIds.filter(id => id !== optionId);
          }
        }
        if ((newData.isError || newData.isWarning) &&
validationIssues.some(issue =>
          issue.id === n.id &&
          (issue.message.includes("не имеет исходящих") ||
            issue.message.includes("не имеет ни одной опции с
действительным переходом") ||
            issue.message.includes("не имеет опций"))
        )) {
```

```
          newData.isError = false;
          newData.isWarning = false;
        }
      }
      if (n.id === params.target && (newData.isError ||
newData.isWarning)) {
        if (validationIssues.some(issue => issue.id === n.id &&
(issue.message.includes("не имеет входящих") ||
issue.message.includes("недостижим")))) {
          nodeChanged = true;
          newData.isError = false;
          newData.isWarning = false;
        }
      }
      return nodeChanged ? { ...n, data: newData } : n;
    }));
  }, [validationIssues, setEdges, setNodes]);
  const updateOptionInStorageAndHandleText = useCallback((optionId,
shortName, dialogueText) => {
    setOptionData(prev => ({ ...prev, [optionId]: { shortName,
dialogueText } }));
    setNodes(nds => nds.map(node => {
      if (node.type === 'statement' && node.data.options?.find(opt
=> opt.id === optionId)) {
        const newErrorOptionIds = (node.data.errorOptionIds ||
[]).filter(id => id !== optionId);
        return {
          ...node, data: {
            ...node.data,
            options: node.data.options.map(opt =>
              opt.id === optionId ? { ...opt, text:
shortName.substring(0, 10) || "..." } : opt
            ),
            errorOptionIds: newErrorOptionIds,
          }
        };
      }
      return node;
    }));
    if (selectedNode && selectedNode.data.options?.find(opt =>
opt.id === optionId)) {
      setSelectedNode(prevSelectedNode => {
        if (!prevSelectedNode) return null;
        const updatedParentNodeFromState = nodes.find(n => n.id ===
prevSelectedNode.id);
        return updatedParentNodeFromState ? {
...updatedParentNodeFromState } : null;
      });
    }
  }, [selectedNode, nodes, validationIssues, setOptionData,
setNodes]);
  const updateOptionHandleAndTarget = useCallback((nodeId,
optionId, newHandleText, newTargetId) => {
```

```
    setNodes((nds) =>
      nds.map((node) => {
        if (node.id === nodeId && node.data.options) {
          const newErrorOptionIds = (node.data.errorOptionIds ||
[]).filter(id => id !== optionId);
          return {
            ...node, data: {
              ...node.data,
              options: node.data.options.map(option =>
                option.id === optionId ? { ...option, text:
newHandleText, target: newTargetId } : option
              ),
              errorOptionIds: newErrorOptionIds,
            }
          };
        }
        return node;
      })
    );
    setEdges(eds => {
      const existingEdge = eds.find(e => e.source === nodeId &&
e.sourceHandle === `bottom-${optionId}`);
      let newEdges = eds.filter(e => !(e.source === nodeId &&
e.sourceHandle === `bottom-${optionId}`));
      if (newTargetId && reactFlowInstance?.getNode(newTargetId)) {
        newEdges.push({
          id: existingEdge?.id || `edge-${uuidv4()}`,
          source: nodeId, target: newTargetId, sourceHandle:
`bottom-${optionId}`,
          targetHandle: `target-default-${newTargetId}`,
          type: 'smoothstep', style: { zIndex: 100, strokeWidth: 2,
stroke: FAT_BORDER_COLOR_APP }
        });
      }
      return newEdges;
    });
    clearErrorForElement(optionId, 'option');
    if (newTargetId) clearErrorForElement(newTargetId, 'node');
  }, [reactFlowInstance, clearErrorForElement, validationIssues,
setNodes, setEdges]);
  const updateStatementData = useCallback((nodeId, shortName,
fullText) => {
    setStatementData(prev => ({ ...prev, [nodeId]: { shortName,
fullText } }));
    setNodes(nds => nds.map(n => (n.id === nodeId ? { ...n, data: {
...n.data, label: shortName, shortName, fullText, isError: false,
isWarning: false } } : n)));
    if (selectedNode?.id === nodeId) {
      setSelectedNode(prevSelectedNode => {
        if (!prevSelectedNode) return null;
        const updatedParentNodeFromState = nodes.find(n => n.id ===
prevSelectedNode.id);
```

```
        return updatedParentNodeFromState ? {
...updatedParentNodeFromState } : null;
      });
    }
  }, [selectedNode, nodes, setStatementData, setNodes]);
  const onNodeClick = useCallback((event, node) => {
    setSelectedNode(node);
    if (selectedNode?.id !== node.id) setSelectedOptionId(null);
  }, [selectedNode]);
  const onNodeDoubleClick = useCallback((event, node) => {
    if (appEditingContext.view === 'metaEditor' && node.type ===
'dialogRef' && onOpenDialogFromMeta) {
      onOpenDialogFromMeta(node.data.dialogId);
    }
  }, [appEditingContext.view, onOpenDialogFromMeta]);
  const onLoad = (_reactFlowInstance) =>
setReactFlowInstance(_reactFlowInstance);
  const onDragOver = useCallback((e) => { e.preventDefault();
e.dataTransfer.dropEffect = 'move'; }, []);
  const onDrop = useCallback((event) => {
    event.preventDefault();
    if (!reactFlowInstance || !reactFlowWrapper.current) return;
    const reactFlowBounds =
reactFlowWrapper.current.getBoundingClientRect();
    const rawData =
event.dataTransfer.getData('application/reactflow');
    let nodeInfo; try { nodeInfo = JSON.parse(rawData); } catch (e)
{ nodeInfo = { type: rawData }; }
    const position = reactFlowInstance.project({ x: event.clientX -
reactFlowBounds.left, y: event.clientY - reactFlowBounds.top });
    const newNodeId = uuidv4();
    let newNode;
    const commonData = { isError: false, isWarning: false };
    switch (nodeInfo.type) {
      case 'start': newNode = { id: newNodeId, type: 'start',
position, data: { label: 'Вход', ...commonData }, style: { zIndex:
10 } }; break;
      case 'end': newNode = { id: newNodeId, type: 'end', position,
data: { label: 'Выход', ...commonData }, style: { zIndex: 10 } };
break;
      case 'statement':
        newNode = { id: newNodeId, type: 'statement', position,
data: { label: 'Высказывание', shortName: 'Высказывание', fullText:
'', options: [], ...commonData }, style: { zIndex: 10 } };
        setStatementData(prev => ({ ...prev, [newNodeId]: {
shortName: 'Высказывание', fullText: '' } }));
        break;
      case 'dialogRef':
        newNode = {
          id: newNodeId, type: 'dialogRef', position, data: {
            label: nodeInfo.label || 'Ссылка', dialogId:
nodeInfo.dialogId,
            entryPoints: nodeInfo.entryPoints || [],
```

```
            exitPoints: nodeInfo.exitPoints || [],
            ...commonData,
          }, style: { zIndex: 10 }
        };
        break;
      default: console.warn("Unknown node type:", nodeInfo.type);
return;
    }
    setNodes((nds) => nds.concat(newNode));
    resetAllErrorHighlighting();
  }, [reactFlowInstance, resetAllErrorHighlighting, setNodes,
setStatementData]);
  const updateNodeLabel = useCallback((nodeId, text) => {
    setNodes((nds) => nds.map(n => (n.id === nodeId && (n.type ===
'start' || n.type === 'end') ? { ...n, data: { ...n.data, label:
text, isError: false, isWarning: false } } : n)));
  }, [setNodes]);
  const handleValidation = useCallback(() => {
    resetAllErrorHighlighting();
    let validationResult;
    if (appEditingContext.view === 'metaEditor') {
      validationResult = validateMetaGraph(nodes, edges,
projectDataForSimulator?.dialogs || []);
    } else {
      validationResult = validateDialogGraph(nodes, edges);
    }
    const currentIssues = validationResult.issues || [];
    setValidationIssues(currentIssues);
    if (currentIssues.length > 0) {
      setIsValidationModalOpen(true);
      const errorNodeIds = new Set(currentIssues.filter(e => e.type
=== 'node' && e.severity === 'error').map(e => e.id));
      const warningNodeIds = new Set(currentIssues.filter(e =>
e.type === 'node' && e.severity === 'warning').map(e => e.id));
      const errorEdgeIds = new Set(currentIssues.filter(e => e.type
=== 'edge' && e.severity === 'error').map(e => e.id));
      const errorOptionDetails = currentIssues.filter(e => e.type
=== 'option' && e.severity === 'error');
      setNodes(nds =>
        nds.map(n => {
          const nodeIsError = errorNodeIds.has(n.id);
          const nodeIsWarning = !nodeIsError &&
warningNodeIds.has(n.id);
          let errorOptionsSet;
          if (n.type === 'statement') {
            errorOptionsSet = new Set(
              errorOptionDetails
                .filter(e => e.contextNodeId === n.id)
                .map(e => e.id)
            );
          }
          const originalBorder = n.style?.originalBorder ||
n.style?.border;
```

```
          return {
            ...n,
            data: {
              ...n.data,
              isError: nodeIsError || (errorOptionsSet &&
errorOptionsSet.size > 0),
              isWarning: nodeIsWarning,
              ...(n.type === 'statement' && { errorOptionIds:
Array.from(errorOptionsSet || []) })
            },
            style: { ...(n.style || {}), originalBorder:
originalBorder }
          };
        })
      );
      setEdges(eds =>
        eds.map(e => {
          const edgeIsError = errorEdgeIds.has(e.id);
          if (edgeIsError) {
            return {
              ...e,
              data: { ...(e.data || {}), originalStroke:
e.style?.stroke, originalStrokeWidth: e.style?.strokeWidth },
              style: { ...(e.style || {}), stroke: '#D32F2F',
strokeWidth: 2.5, }
            };
          }
          return e;
        })
      );
    } else {
      alert("Проверка пройдена успешно! Замечаний нет.");
    }
  }, [nodes, edges, resetAllErrorHighlighting,
appEditingContext.view, projectDataForSimulator, setNodes,
setEdges, setValidationIssues, setIsValidationModalOpen]);
  const addOption = useCallback((nodeId) => {
    const newOptionId = uuidv4();
    let newOptionText = "Реакция1";
    setNodes((nds) => nds.map(n => {
      if (n.id === nodeId && n.type === 'statement') {
        const optionIndex = (n.data.options || []).length + 1;
        newOptionText = `Реакция${optionIndex}`;
        const wasMissingOptions = validationIssues.some(issue =>
issue.id === nodeId && issue.message.includes("не имеет опций"));
        return {
          ...n,
          data: {
            ...n.data,
            options: [...(n.data.options || []), { id: newOptionId,
text: newOptionText, target: '' }],
            isError: wasMissingOptions ? false : n.data.isError,
```

```
            isWarning: wasMissingOptions ? false :
n.data.isWarning,
          }
        };
      } return n;
    }));
    setOptionData(prev => ({ ...prev, [newOptionId]: { shortName:
newOptionText, dialogueText: '' } }));
    if (validationIssues.some(issue => issue.id === nodeId &&
issue.message.includes("не имеет опций"))) {
      clearErrorForElement(nodeId, 'node');
    }
  }, [validationIssues, clearErrorForElement, setNodes,
setOptionData]);
  const onConnectStart = useCallback((_, { handleType }) => { if
(handleType === 'source') setIsConnecting(true); }, []);
  const onConnectEnd = useCallback(() => setIsConnecting(false),
[]);
  const defaultEdgeOptionsAll = {
    style: { stroke: FAT_BORDER_COLOR_APP, strokeWidth: 2, zIndex:
100 },
    markerEnd: { type: 'arrowclosed', width: 15, height: 15, color:
FAT_BORDER_COLOR_APP, strokeWidth: 1 },
    type: 'smoothstep',
    zIndex: 100,
  };
  const appContainerStyle = {
    height: '100%',
    display: 'flex',
    flexDirection: 'column',
    background: LIGHT_BACKGROUND_APP,
    fontFamily: FONT_FAMILY_TEXT_APP
  };
  const topBarStyle = {
    background: LIGHT_BACKGROUND_APP,
    borderBottom: `${BORDER_WIDTH_APP} solid
${FAT_BORDER_COLOR_APP}`,
    padding: '8px 12px',
    display: 'flex',
    alignItems: 'center',
    gap: '8px',
    zIndex: 200,
    flexShrink: 0
  };
  const topBarButtonStyle = (isActive = false, isSpecial = false)
=> ({
    background: isActive ? DARK_TEXT_COLOR_APP : (isSpecial ?
ACCENT_COLOR_PRIMARY_APP : '#FFFFFF'),
    color: isActive ? LIGHT_BACKGROUND_APP : (isSpecial ? '#FFFFFF'
: DARK_TEXT_COLOR_APP),
    border: `${BORDER_WIDTH_APP} solid ${FAT_BORDER_COLOR_APP}`,
    padding: '6px 10px',
    borderRadius: '0px',
```

```
    cursor: 'pointer',
    fontSize: '0.9em',
    fontWeight: 'bold',
    transition: 'transform 0.07s, box-shadow 0.07s, background-
color 0.07s, color 0.07s',
    outline: 'none',
    whiteSpace: 'nowrap',
    fontFamily: FONT_FAMILY_HEADINGS_APP,
    textTransform: 'uppercase',
    letterSpacing: '0.5px',
    boxShadow: BUTTON_SHADOW_APP,
    transform: BUTTON_TRANSFORM_INITIAL_APP,
  });
  const topBarButtonHoverStyle = (isSpecial = false) => ({
    background: isSpecial ? '#005c9e' : DARK_TEXT_COLOR_APP,
    color: LIGHT_BACKGROUND_APP,
    boxShadow: BUTTON_SHADOW_HOVER_APP,
    transform: BUTTON_TRANSFORM_HOVER_APP,
  });
  const topBarButtonActiveStyle = (isSpecial = false) => ({
    background: isSpecial ? '#004272' : '#000000',
    color: LIGHT_BACKGROUND_APP,
    boxShadow: BUTTON_SHADOW_ACTIVE_APP,
    transform: BUTTON_TRANSFORM_ACTIVE_APP,
  });
  const dialogNameStyle = {
    fontSize: '1.1em',
    fontWeight: 'bold',
    fontFamily: FONT_FAMILY_HEADINGS_APP,
    color: DARK_TEXT_COLOR_APP,
    marginLeft: 'auto',
    marginRight: '15px',
    whiteSpace: 'nowrap',
    overflow: 'hidden',
    textOverflow: 'ellipsis',
    maxWidth: '300px',
    borderBottom: `${BORDER_WIDTH_APP} dotted
${FAT_BORDER_COLOR_APP}`,
    paddingBottom: '3px',
  };
  const dropdownMenuContainerStyle = { position: 'relative' };
  const dropdownMenuStyle = {
    position: 'absolute',
    top: 'calc(100% + 4px)',
    left: 0,
    background: LIGHT_BACKGROUND_APP,
    border: `${BORDER_WIDTH_APP} solid ${FAT_BORDER_COLOR_APP}`,
    borderRadius: '0px',
    boxShadow: `3px 3px 0px ${FAT_BORDER_COLOR_APP}`,
    zIndex: 1001,
    minWidth: '220px',
    padding: '5px 0'
  };
```

```
  const dropdownButtonStyle = {
    display: 'block',
    width: '100%',
    textAlign: 'left',
    padding: '8px 12px',
    border: 'none',
    background: 'none',
    cursor: 'pointer',
    fontSize: '0.9em',
    fontWeight: 'bold',
    color: DARK_TEXT_COLOR_APP,
    fontFamily: FONT_FAMILY_TEXT_APP,
    transition: 'background-color 0.05s, color 0.05s',
  };
  const dropdownButtonHoverStyle = {
    backgroundColor: DARK_TEXT_COLOR_APP,
    color: LIGHT_BACKGROUND_APP,
  };
  const editorContentWrapperStyle = { display: 'flex', flexGrow: 1,
overflow: 'hidden' };
  const reactFlowWrapperStyle = {
    flexGrow: 1,
    height: '100%',
    background: '#EAEAEA',
    position: 'relative'
  };
  const controlsStyle = {
    position: 'absolute',
    left: '15px',
    bottom: '15px',
    background: LIGHT_BACKGROUND_APP,
    border: `${BORDER_WIDTH_APP} solid ${FAT_BORDER_COLOR_APP}`,
    borderRadius: '0px',
    boxShadow: `2px 2px 0px ${FAT_BORDER_COLOR_APP}`,
    padding: '5px',
    zIndex: 200
  };
  const handleOpenSimulator = () => {
    let initialSimNodeId = null;
    let initialDialogEntryPoints = null;
    if (appEditingContext.view === 'dialogEditor') {
        initialSimNodeId = nodes.find(n => n.type === 'start')?.id;
    } else if (appEditingContext.view === 'metaEditor') {
        const potentialStartNodes = nodes.filter(n =>
            n.type === 'start' ||
            (n.type === 'dialogRef' && edges.filter(e => e.target
=== n.id).length === 0)
        );
        if (potentialStartNodes.length > 0) {
            const firstStartNode = potentialStartNodes.sort((a,b)
=> (a.type === 'start' ? -1 : (b.type === 'start' ? 1 : 0)))[0];
            initialSimNodeId = firstStartNode.id;
```

```
            if (firstStartNode.type === 'dialogRef' &&
projectDataForSimulator) {
                const dialogId = firstStartNode.data.dialogId;
                const dialogData =
projectDataForSimulator.dialogs.find(d => d.id === dialogId);
                if (dialogData) {
                    const deserializedDialogNodes =
deserializeNodes(dialogData.nodes);
                    initialDialogEntryPoints =
getDialogEntryPoints(deserializedDialogNodes);
                     if (initialDialogEntryPoints?.length === 1) {
                    } else if (initialDialogEntryPoints?.length ===
0 && firstStartNode.data.entryPoints?.length > 0) {
                        initialDialogEntryPoints =
firstStartNode.data.entryPoints;
                    }
                }
            }
        }
    }
    setSimulatorInitialProps({ nodeId: initialSimNodeId,
entryPoints: initialDialogEntryPoints });
    setIsSimulatorOpen(true);
  };
  const handleCloseSimulator = () => setIsSimulatorOpen(false);
  return (
    <div className="dndflow" style={appContainerStyle}>
      <div style={topBarStyle}>
        <button onClick={handleBackToProjectView}
style={topBarButtonStyle(false, true)}
          onMouseEnter={e => Object.assign(e.currentTarget.style,
topBarButtonHoverStyle(true))}
          onMouseLeave={e => Object.assign(e.currentTarget.style,
topBarButtonStyle(showSaveDialogioMenu, true))}
          onMouseDown={e => Object.assign(e.currentTarget.style,
topBarButtonActiveStyle(true))}
          onMouseUp={e => Object.assign(e.currentTarget.style,
topBarButtonHoverStyle(true))}
          >
          Назад в Проект
        </button>
        <div style={dropdownMenuContainerStyle}>
            <button onClick={() => setShowSaveDialogioMenu(prev =>
!prev)} style={topBarButtonStyle(showSaveDialogioMenu)}
                onMouseEnter={e =>
Object.assign(e.currentTarget.style, topBarButtonHoverStyle())}
                onMouseLeave={e =>
Object.assign(e.currentTarget.style,
topBarButtonStyle(showSaveDialogioMenu))}
                onMouseDown={e =>
Object.assign(e.currentTarget.style, topBarButtonActiveStyle())}
                onMouseUp={e =>
Object.assign(e.currentTarget.style, topBarButtonHoverStyle())}
```

```
                >
                .dialogio
            </button>
            {showSaveDialogioMenu && (
            <div style={dropdownMenuStyle}>
                <button onClick={exportGraphToDialogio}
style={dropdownButtonStyle}
                    onMouseEnter={e =>
Object.assign(e.currentTarget.style, dropdownButtonHoverStyle)}
                    onMouseLeave={e =>
Object.assign(e.currentTarget.style,
dropdownButtonStyle)}>Сохранить как .dialogio</button>
                <label htmlFor="import-file-dialogio-app"
style={{...dropdownButtonStyle, display: 'block'}}
                    onMouseEnter={e =>
Object.assign(e.currentTarget.style, dropdownButtonHoverStyle)}
                    onMouseLeave={e =>
Object.assign(e.currentTarget.style, dropdownButtonStyle)}>Открыть
.dialogio
                <input type="file" id="import-file-dialogio-app"
accept=".dialogio" onChange={importGraphFromDialogio} style={{
display: 'none' }}/>
                </label>
            </div>
            )}
        </div>
        <div style={dropdownMenuContainerStyle}>
            <button onClick={() => setShowExportMenu(prev =>
!prev)} style={topBarButtonStyle(showExportMenu)}
                onMouseEnter={e =>
Object.assign(e.currentTarget.style, topBarButtonHoverStyle())}
                onMouseLeave={e =>
Object.assign(e.currentTarget.style,
topBarButtonStyle(showExportMenu))}
                onMouseDown={e =>
Object.assign(e.currentTarget.style, topBarButtonActiveStyle())}
                onMouseUp={e =>
Object.assign(e.currentTarget.style, topBarButtonHoverStyle())}
                >
                Экспорт / Импорт
            </button>
            {showExportMenu && (
            <div style={dropdownMenuStyle}>
                <button onClick={exportGraphToJson}
style={dropdownButtonStyle} onMouseEnter={e =>
Object.assign(e.currentTarget.style, dropdownButtonHoverStyle)}
onMouseLeave={e => Object.assign(e.currentTarget.style,
dropdownButtonStyle)}>Экспорт в .json</button>
                <label htmlFor="import-file-json-app"
style={{...dropdownButtonStyle, display: 'block'}} onMouseEnter={e
=> Object.assign(e.currentTarget.style, dropdownButtonHoverStyle)}
onMouseLeave={e => Object.assign(e.currentTarget.style,
dropdownButtonStyle)}>Импорт из .json
```

```
                <input type="file" id="import-file-json-app"
accept=".json" onChange={importGraphFromJson} style={{ display:
'none' }}/>
                </label>
                <hr style={{margin: '4px 0', border: 'none',
borderTop: `1px solid ${FAT_BORDER_COLOR_APP}`}}/>
                <button onClick={() => exportToImage('png')}
style={dropdownButtonStyle} onMouseEnter={e =>
Object.assign(e.currentTarget.style, dropdownButtonHoverStyle)}
onMouseLeave={e => Object.assign(e.currentTarget.style,
dropdownButtonStyle)}>Экспорт в PNG</button>
                <button onClick={() => exportToImage('jpeg')}
style={dropdownButtonStyle} onMouseEnter={e =>
Object.assign(e.currentTarget.style, dropdownButtonHoverStyle)}
onMouseLeave={e => Object.assign(e.currentTarget.style,
dropdownButtonStyle)}>Экспорт в JPG</button>
            </div>
            )}
        </div>
         <button onClick={handleValidation}
style={topBarButtonStyle()}
          onMouseEnter={e => Object.assign(e.currentTarget.style,
topBarButtonHoverStyle())}
          onMouseLeave={e => Object.assign(e.currentTarget.style,
topBarButtonStyle())}
          onMouseDown={e => Object.assign(e.currentTarget.style,
topBarButtonActiveStyle())}
          onMouseUp={e => Object.assign(e.currentTarget.style,
topBarButtonHoverStyle())}
          >
            Проверка
        </button>
        <button onClick={handleOpenSimulator}
style={topBarButtonStyle()}
            onMouseEnter={e => Object.assign(e.currentTarget.style,
topBarButtonHoverStyle())}
            onMouseLeave={e => Object.assign(e.currentTarget.style,
topBarButtonStyle())}
            onMouseDown={e => Object.assign(e.currentTarget.style,
topBarButtonActiveStyle())}
            onMouseUp={e => Object.assign(e.currentTarget.style,
topBarButtonHoverStyle())}
            >
            Симулятор
        </button>
        <span style={dialogNameStyle}
title={currentName}>{currentName}</span>
      </div>
      <ReactFlowProvider>
        <div style={editorContentWrapperStyle}>
          {sidebarToRender}
          <div className="reactflow-wrapper" ref={reactFlowWrapper}
style={reactFlowWrapperStyle}>
```

```
          <ReactFlow
              nodes={nodes} edges={edges}
              onNodesChange={onNodesChange}
onEdgesChange={onEdgesChange}
              onConnect={onConnect} onConnectStart={onConnectStart}
onConnectEnd={onConnectEnd}
              onInit={onLoad} onDrop={onDrop}
onDragOver={onDragOver}
              onNodeClick={onNodeClick}
onNodeDoubleClick={onNodeDoubleClick}
              nodeTypes={memoizedNodeTypes}
defaultEdgeOptions={defaultEdgeOptionsAll}
              fitView fitViewOptions={{ padding: 0.15, duration:
300 }}
              deleteKeyCode={['Backspace', 'Delete']}
              elementsSelectable={true} nodesDraggable={true}
nodesConnectable={true}
              attributionPosition={null}
              elevateEdgesOnSelect={true}
            >
              <Background color="#CCCCCC" gap={25} size={1}
variant="dots" /> {}
              <Controls style={controlsStyle}
showInteractive={false} />
           </ReactFlow>
          </div>
          <NodeEditor
              key={selectedNode?.id || 'no-selection'}
              node={selectedNode}
              onUpdate={updateNodeLabel}
              onAddOption={addOption}
              onUpdateOption={updateOptionHandleAndTarget}
              optionDataStorage={optionData}

updateOptionInStorage={updateOptionInStorageAndHandleText}
              onOptionDelete={deleteOption}
              onNodeDelete={deleteNode}
              statementDataStorage={statementData}
              updateStatementData={updateStatementData}
              setSelectedOptionId={setSelectedOptionId}
              selectedOptionId={selectedOptionId}
              clearErrorForElement={clearErrorForElement}
            />
        </div>
      </ReactFlowProvider>
      {isSimulatorOpen && (
        <DialogSimulatorModal
          isOpen={isSimulatorOpen}
          onRequestClose={handleCloseSimulator}
          nodes={nodes}
          edges={edges}
          statementData={statementData}
          optionData={optionData}
```

```
          isMetaGraph={appEditingContext.view === 'metaEditor'}
          projectData={appEditingContext.view === 'metaEditor' ?
projectDataForSimulator : null}
          initialNodeIdProp={simulatorInitialProps.nodeId}

initialEntryPointsProp={simulatorInitialProps.entryPoints}
        />
      )}
      <ValidationErrorsModal
        isOpen={isValidationModalOpen}
        onRequestClose={() => {
            setIsValidationModalOpen(false);
        }}
        issues={validationIssues}
      />
    </div>
  );
});
export default App;
```

Листинг Б.2 – Файл NodeEditor.jsx

```
import React, { useState, useEffect, useRef, useCallback } from
'react';
import Modal from 'react-modal';
const LIGHT_BACKGROUND = '#F5F5F5';
const DARK_TEXT_COLOR = '#1E1E1E';
const FAT_BORDER_COLOR = '#1A1A1A';
const BORDER_WIDTH = '2px';
const ACCENT_COLOR_PRIMARY = '#007ACC';
const ACCENT_COLOR_SUCCESS = '#4CAF50';
const ACCENT_COLOR_DANGER = '#F44336';
const FONT_FAMILY_HEADINGS = "'Arial Black', 'Impact', sans-serif";
const FONT_FAMILY_TEXT = "'Arial', sans-serif";
const BUTTON_SHADOW = `3px 3px 0px ${FAT_BORDER_COLOR}`;
const BUTTON_TRANSFORM_INITIAL = 'translate(-1px, -1px)';
const BUTTON_SHADOW_HOVER = `1px 1px 0px ${FAT_BORDER_COLOR}`;
const BUTTON_TRANSFORM_HOVER = 'translate(0px, 0px)';
const BUTTON_SHADOW_ACTIVE = `0px 0px 0px ${FAT_BORDER_COLOR}`;
const BUTTON_TRANSFORM_ACTIVE = 'translate(1px, 1px)';
const INPUT_FOCUS_BORDER_COLOR = ACCENT_COLOR_PRIMARY;
const NodeEditor = ({
  node,
  onUpdate,
  statementDataStorage,
  updateStatementData,
  onAddOption,
  onUpdateOption,
  optionDataStorage,
  updateOptionInStorage,
  onOptionDelete,
  onNodeDelete,
  selectedOptionId,
  setSelectedOptionId,
```

```
  clearErrorForElement,
}) => {
  const [isEditing, setIsEditing] = useState(false);
  const [tempLabel, setTempLabel] = useState('');
  const [statementModalIsOpen, setStatementModalIsOpen] =
useState(false);
  const [tempStmtShortName, setTempStmtShortName] = useState('');
  const [tempStmtFullText, setTempStmtFullText] = useState('');
  const [isEditingLabel, setIsEditingLabel] = useState(false);
  const [labelText, setLabelText] = useState((node?.data?.label ||
"").substring(0, 800));
  const [stmtShortName, setStmtShortName] =
useState((node?.data?.shortName ||
statementDataStorage[node?.id]?.shortName || "").substring(0, 70));
  const [stmtFullText, setStmtFullText] =
useState((node?.data?.fullText ||
statementDataStorage[node?.id]?.fullText || "").substring(0,
2000));
  const [isInlineEditingStatement, setIsInlineEditingStatement] =
useState(false);
  const [isResizing, setIsResizing] = useState(false);
  const [editorWidth, setEditorWidth] = useState(450);
  const minWidth = 380;
  const maxWidth = 1000;
  const handleAnyChange = () => {
    if (node?.data?.isError || node?.data?.isWarning) {
        clearErrorForElement(node.id, 'node');
    }
  };
  useEffect(() => {
    setLabelText((node?.data?.label || "").substring(0, 800));
    setStmtShortName((node?.data?.shortName ||
statementDataStorage[node?.id]?.shortName || "").substring(0, 70));
    setStmtFullText((node?.data?.fullText ||
statementDataStorage[node?.id]?.fullText || "").substring(0,
2000));
    setIsEditingLabel(false);
    setIsInlineEditingStatement(false);
  }, [node, statementDataStorage]);
  const handleLabelChange = (e) => {
    setLabelText(e.target.value.substring(0, 800));
    handleAnyChange();
  };
  const handleSaveLabel = () => {
    if (node && node.type !== 'statement') {
      onUpdate(node.id, labelText.substring(0, 800));
    }
    setIsEditingLabel(false);
  };
  const handleStmtShortNameChange = (e) => {
    setStmtShortName(e.target.value.substring(0, 70));
    handleAnyChange();
  };
```

```
  const handleStmtFullTextChange = (e) => {
    setStmtFullText(e.target.value.substring(0, 2000));
    handleAnyChange();
  };
  const handleSaveStatementInline = () => {
    updateStatementData(node.id, stmtShortName.substring(0, 70),
stmtFullText.substring(0, 2000));
    setIsInlineEditingStatement(false);
  };
  const handleCancelStatementInline = () => {
    setStmtShortName((node?.data?.shortName ||
statementDataStorage[node?.id]?.shortName || "").substring(0, 70));
    setStmtFullText((node?.data?.fullText ||
statementDataStorage[node?.id]?.fullText || "").substring(0,
2000));
    setIsInlineEditingStatement(false);
  };
  const handleOpenStatementModal = () => {
    setStmtShortName((node?.data?.shortName ||
statementDataStorage[node?.id]?.shortName || "").substring(0, 70));
    setStmtFullText((node?.data?.fullText ||
statementDataStorage[node?.id]?.fullText || "").substring(0,
2000));
    setStatementModalIsOpen(true);
  };
  const handleCloseStatementModal = () =>
setStatementModalIsOpen(false);
  const handleSaveStatementModal = () => {
    updateStatementData(node.id, stmtShortName.substring(0, 70),
stmtFullText.substring(0, 2000));
    setStatementModalIsOpen(false);
  };
  const handleDeleteNode = () => {
    if (window.confirm(`Удалить узел "${node.data.label ||
node.data.shortName || node.id}"?`)) {
      onNodeDelete(node.id);
    }
  };
  const resizerRef = useRef(null);
  const handleMouseDownResize = useCallback((e) => {
e.preventDefault(); setIsResizing(true); }, []);
  const handleMouseMoveResize = useCallback((e) => {
    if (!isResizing) return;
    const newWidth = window.innerWidth - e.clientX;
    if (newWidth >= minWidth && newWidth <= maxWidth)
setEditorWidth(newWidth);
  }, [isResizing, minWidth, maxWidth]);
  const handleMouseUpResize = useCallback(() => {
setIsResizing(false); }, []);
  useEffect(() => {
    if (isResizing) {
      document.addEventListener('mousemove',
handleMouseMoveResize);
```

```
      document.addEventListener('mouseup', handleMouseUpResize);
    } else {
      document.removeEventListener('mousemove',
handleMouseMoveResize);
      document.removeEventListener('mouseup', handleMouseUpResize);
    }
    return () => {
      document.removeEventListener('mousemove',
handleMouseMoveResize);
      document.removeEventListener('mouseup', handleMouseUpResize);
    };
  }, [isResizing, handleMouseMoveResize, handleMouseUpResize]);
  const editorStyle = {
    width: `${editorWidth}px`,
    minWidth: `${minWidth}px`,
    maxWidth: `${maxWidth}px`,
    background: LIGHT_BACKGROUND,
    borderLeft: `${BORDER_WIDTH} solid ${FAT_BORDER_COLOR}`,
    display: 'flex',
    flexDirection: 'column',
    height: '100%',
    overflow: 'hidden',
    position: 'relative',
    boxSizing: 'border-box',
    fontFamily: FONT_FAMILY_TEXT,
    color: DARK_TEXT_COLOR,
  };
  const editorHeaderStyle = {
    padding: '15px 20px',
    borderBottom: `${BORDER_WIDTH} solid ${FAT_BORDER_COLOR}`,
    flexShrink: 0,
    background: '#FFFFFF',
  };
  const editorContentStyle = {
    padding: '20px',
    overflowY: 'auto',
    flexGrow: 1
  };
  const editorFooterStyle = {
    padding: '15px 20px',
    borderTop: `${BORDER_WIDTH} solid ${FAT_BORDER_COLOR}`,
    flexShrink: 0,
    background: '#FFFFFF',
  };
  const resizerStyle = {
    position: 'absolute',
    left: `-${parseInt(BORDER_WIDTH,10)/2 + 1}px`,
    top: 0, bottom: 0, width: '8px',
    cursor: 'ew-resize', zIndex: 1,
    background: isResizing ? ACCENT_COLOR_PRIMARY : 'transparent'
  };
  const sectionStyle = {
    background: '#FFFFFF',
```

```
    padding: '15px',
    borderRadius: '0px',
    border: `${BORDER_WIDTH} solid ${FAT_BORDER_COLOR}`,
    marginBottom: '20px',
    boxShadow: BUTTON_SHADOW,
  };
  const headingStyle = (level = 3) => ({
    color: DARK_TEXT_COLOR,
    fontSize: level === 3 ? '1.3em' : '1.1em',
    fontWeight: '900',
    fontFamily: FONT_FAMILY_HEADINGS,
    borderBottom: `${BORDER_WIDTH} solid ${FAT_BORDER_COLOR}`,
    paddingBottom: '8px',
    marginBottom: '15px',
    textTransform: 'uppercase',
    letterSpacing: '0.5px',
  });
  const labelStyle = {
    display: 'block',
    marginBottom: '8px',
    color: DARK_TEXT_COLOR,
    fontSize: '1em',
    fontWeight: 'bold',
    fontFamily: FONT_FAMILY_TEXT,
  };
  const inputBaseStyle = {
    width: '100%',
    padding: '10px',
    border: `${BORDER_WIDTH} solid ${FAT_BORDER_COLOR}`,
    borderRadius: '0px',
    fontSize: '1em',
    boxSizing: 'border-box',
    marginBottom: '12px',
    outline: 'none',
    background: LIGHT_BACKGROUND,
    color: DARK_TEXT_COLOR,
    fontFamily: FONT_FAMILY_TEXT,
    boxShadow: BUTTON_SHADOW,
    transform: BUTTON_TRANSFORM_INITIAL,
    transition: 'border-color 0.07s, box-shadow 0.07s, transform
0.07s, background-color 0.07s',
  };
  const inputFocusStyle = {
    borderColor: INPUT_FOCUS_BORDER_COLOR,
    background: '#FFFFFF',
    boxShadow: BUTTON_SHADOW_HOVER,
    transform: BUTTON_TRANSFORM_HOVER,
  };
  const textareaStyle = {
    ...inputBaseStyle,
    minHeight: '100px',
    resize: 'vertical',
    lineHeight: 1.5
```

```
  };
  const brutalistButton = (type = 'default', fullWidth = false) =>
{
    let bg = '#FFFFFF';
    let color = DARK_TEXT_COLOR;
    let hoverBg = DARK_TEXT_COLOR;
    let hoverColor = LIGHT_BACKGROUND;
    let activeBg = '#000000';
    let borderColor = FAT_BORDER_COLOR;
    if (type === 'primary') { bg = ACCENT_COLOR_PRIMARY; color =
'#FFFFFF'; hoverBg = '#005c9e'; activeBg = '#004272'; borderColor =
FAT_BORDER_COLOR;}
    else if (type === 'success') { bg = ACCENT_COLOR_SUCCESS; color
= '#FFFFFF'; hoverBg = '#3e8e41'; activeBg = '#2d6a30'; borderColor
= FAT_BORDER_COLOR;}
    else if (type === 'danger') { bg = ACCENT_COLOR_DANGER; color =
'#FFFFFF'; hoverBg = '#d43f3a'; activeBg = '#b0302c'; borderColor =
FAT_BORDER_COLOR;}
    return {
      background: bg, color: color,
      borderColor: borderColor,
      borderWidth: BORDER_WIDTH,
      borderStyle: 'solid',
      padding: '10px 15px',
      borderRadius: '0px',
      cursor: 'pointer',
      fontSize: '1em',
      fontWeight: 'bold',
      fontFamily: FONT_FAMILY_HEADINGS,
      transition: 'transform 0.07s ease-out, box-shadow 0.07s ease-
out, background-color 0.07s ease-out, color 0.07s ease-out',
      outline: 'none',
      textTransform: 'uppercase',
      letterSpacing: '0.5px',
      width: fullWidth ? '100%' : 'auto',
      boxShadow: BUTTON_SHADOW,
      transform: BUTTON_TRANSFORM_INITIAL,
      _hover: {
        background: hoverBg,
        color: hoverColor,
        boxShadow: BUTTON_SHADOW_HOVER,
        transform: BUTTON_TRANSFORM_HOVER,
      },
      _active: {
        background: activeBg,
        color: hoverColor,
        boxShadow: BUTTON_SHADOW_ACTIVE,
        transform: BUTTON_TRANSFORM_ACTIVE,
      }
    };
  };
  const getButtonDynamicStyles = (buttonType = 'default',
isFullWidth = false) => {
```

```
    const base = brutalistButton(buttonType, isFullWidth);
    return {
      base: base,
      hover: {...base, ...base._hover},
      active: {...base, ...base._active}
    };
  };
  const btnDefaultStyles = getButtonDynamicStyles('default');
  const btnPrimaryStyles = getButtonDynamicStyles('primary');
  const btnSuccessStyles = getButtonDynamicStyles('success');
  const btnDangerStyles = getButtonDynamicStyles('danger');
  const hrStyle = { border: 'none', borderTop: `${BORDER_WIDTH}
solid ${FAT_BORDER_COLOR}`, margin: '25px 0' };
  const noNodeSelectedContainerStyle = {
    display: 'flex',
    flexDirection: 'column',
    alignItems: 'center',
    justifyContent: 'center',
    height: '100%',
    textAlign: 'center',
    padding: '20px',
    background: LIGHT_BACKGROUND,
  };
  const noNodeSelectedTextStyle = {
    fontFamily: FONT_FAMILY_HEADINGS,
    fontSize: '1.5em',
    color: DARK_TEXT_COLOR,
    textTransform: 'uppercase',
    letterSpacing: '1px',
    marginBottom: '10px',
  };
  const noNodeSelectedSubTextStyle = {
    fontFamily: FONT_FAMILY_TEXT,
    fontSize: '1.1em',
    color: DARK_TEXT_COLOR,
    opacity: 0.8,
  };
  if (!node) {
    return (
      <div style={{
        width: '300px',
        padding: '20px',
        background: '#FFFFFF',
        borderLeft: '2px solid #1A1A1A',
        display: 'flex',
        flexDirection: 'column',
        gap: '15px',
        height: '100%',
        overflowY: 'auto'
      }}>
        <h3 style={{
          fontSize: '1.2em',
          fontWeight: 'bold',
```

```
          fontFamily: "'Arial Black', 'Impact', sans-serif",
          color: '#1A1A1A',
          margin: 0,
          textAlign: 'center'
        }}>
          Выберите узел
        </h3>
      </div>
    );
  }
  return (
    <div style={editorStyle}>
      <div style={resizerStyle} ref={resizerRef}
onMouseDown={handleMouseDownResize}
        onMouseEnter={e => e.currentTarget.style.background =
'rgba(0,0,0,0.1)'}
        onMouseLeave={e => e.currentTarget.style.background =
isResizing ? ACCENT_COLOR_PRIMARY : 'transparent'}
      />
      <div style={editorHeaderStyle}>
        <h3 style={{ ...headingStyle(), borderBottom: 'none',
margin: 0, fontSize: '1.4em' }}>{node.data?.label ||
node.data?.shortName || `Узел: ${node.id.substring(0,8)}`}</h3>
        <div style={{fontSize: '0.85em', color: '#666', marginTop:
'5px', fontFamily: FONT_FAMILY_TEXT}}>ID: {node.id} | Тип:
{node.type}</div>
      </div>
      <div style={editorContentStyle}>
        {node.type !== 'statement' && node.type !== 'dialogRef' &&
(
          <div style={sectionStyle}>
            <h4 style={headingStyle(4)}>Название узла</h4>
            {isEditingLabel ? (
              <>
                <input type="text" value={labelText}
                  onChange={handleLabelChange}
                  maxLength={800}
                  style={inputBaseStyle} autoFocus
                  onFocus={e => Object.assign(e.target.style,
inputFocusStyle)}
                  onBlur={e => Object.assign(e.target.style,
inputBaseStyle)}
                />
                <div style={{display: 'flex', gap: '10px',
marginTop: '5px'}}>
                    <button
                        onClick={handleSaveLabel}
                        style={btnSuccessStyles.base}
                        onMouseEnter={e =>
Object.assign(e.currentTarget.style, btnSuccessStyles.hover)}
                        onMouseLeave={e =>
Object.assign(e.currentTarget.style, btnSuccessStyles.base)}
```

```
                        onMouseDown={e =>
Object.assign(e.currentTarget.style, btnSuccessStyles.active)}
                        onMouseUp={e =>
Object.assign(e.currentTarget.style, btnSuccessStyles.hover)}
                    >Сохранить</button>
                    <button
                        onClick={() => { setIsEditingLabel(false);
setLabelText((node.data.label || "").substring(0,800)); }}
                        style={btnDefaultStyles.base}
                        onMouseEnter={e =>
Object.assign(e.currentTarget.style, btnDefaultStyles.hover)}
                        onMouseLeave={e =>
Object.assign(e.currentTarget.style, btnDefaultStyles.base)}
                        onMouseDown={e =>
Object.assign(e.currentTarget.style, btnDefaultStyles.active)}
                        onMouseUp={e =>
Object.assign(e.currentTarget.style, btnDefaultStyles.hover)}
                    >Отмена</button>
          </div>
              </>
            ) : (
              <>
                <p style={{ margin: '0 0 15px 0', padding: '10px',
background: LIGHT_BACKGROUND, borderRadius: '0px', border:
`${BORDER_WIDTH} solid ${FAT_BORDER_COLOR}`, fontFamily:
FONT_FAMILY_TEXT }}>{labelText || '(без названия)'}</p>
                <button onClick={() => setIsEditingLabel(true)}
                    style={btnPrimaryStyles.base}
                    onMouseEnter={e =>
Object.assign(e.currentTarget.style, btnPrimaryStyles.hover)}
                    onMouseLeave={e =>
Object.assign(e.currentTarget.style, btnPrimaryStyles.base)}
                    onMouseDown={e =>
Object.assign(e.currentTarget.style, btnPrimaryStyles.active)}
                    onMouseUp={e =>
Object.assign(e.currentTarget.style, btnPrimaryStyles.hover)}
                >Изменить</button>
              </>
            )}
          </div>
        )}
         {node.type === 'dialogRef' && (
          <>
            <div style={sectionStyle}>
              <h4 style={headingStyle(4)}>Информация о ссылке на
диалог</h4>
              <p style={{fontFamily: FONT_FAMILY_TEXT}}><strong>ID
диалога:</strong> {node.data.dialogId}</p>
              <p style={{fontFamily:
FONT_FAMILY_TEXT}}><strong>Метка (имя диалога):</strong>
{node.data.label}</p>
            </div>
            <div style={sectionStyle}>
```

```
              <h4 style={headingStyle(4)}>Точки входа диалога</h4>
              {node.data.entryPoints &&
node.data.entryPoints.length > 0 ? (
                <ul style={{ listStyle: 'none', padding: 0, margin:
0, fontFamily: FONT_FAMILY_TEXT }}>
                  {node.data.entryPoints.map(point => (
                    <li key={`entry-${point.id}`} style={{ padding:
'4px 0', fontSize: '0.9em', borderBottom: `1px solid
${FAT_BORDER_COLOR}` }}>
                      <strong>ID:</strong>
{point.id.substring(0,8)}... <br/>
                      <strong>Метка:</strong> {point.label || '(без
метки)'}
                    </li>
                  ))}
                </ul>
              ) : (
                <p style={{color: DARK_TEXT_COLOR, opacity:0.7,
fontFamily: FONT_FAMILY_TEXT}}>Нет определенных точек входа.</p>
              )}
            </div>
              <div style={sectionStyle}>
                <h4 style={headingStyle(4)}>Точки выхода диалога</h4>
                {node.data.exitPoints && node.data.exitPoints.length
> 0 ? (
                  <ul style={{ listStyle: 'none', padding: 0, margin:
0, fontFamily: FONT_FAMILY_TEXT }}>
                    {node.data.exitPoints.map(point => (
                      <li key={`exit-${point.id}`} style={{ padding:
'4px 0', fontSize: '0.9em', borderBottom: `1px solid
${FAT_BORDER_COLOR}` }}>
                        <strong>ID:</strong>
{point.id.substring(0,8)}... <br/>
                        <strong>Метка:</strong> {point.label || '(без
метки)'}
                      </li>
                    ))}
                  </ul>
              ) : (
                  <p style={{color: DARK_TEXT_COLOR, opacity:0.7,
fontFamily: FONT_FAMILY_TEXT}}>Нет определенных точек выхода.</p>
                )}
              </div>
            </>
          )}
          {node.type === 'statement' && (
            <div style={sectionStyle}>
              <div style={{display: 'flex', justifyContent: 'space-
between', alignItems: 'center', marginBottom: '15px'}}>
                <h4 style={{...headingStyle(4), margin:0, border:0,
padding:0}}>Данные высказывания</h4>
                <button onClick={handleOpenStatementModal}
```

```
style={{...getButtonDynamicStyles('default').base, padding: '5px
10px', fontSize: '0.8em'}}
                    onMouseEnter={e =>
Object.assign(e.currentTarget.style,
getButtonDynamicStyles('default').hover)}
                    onMouseLeave={e =>
Object.assign(e.currentTarget.style,
getButtonDynamicStyles('default').base)}
                    onMouseDown={e =>
Object.assign(e.currentTarget.style,
getButtonDynamicStyles('default').active)}
                    onMouseUp={e =>
Object.assign(e.currentTarget.style,
getButtonDynamicStyles('default').hover)}
                    title="Открыть в модальном окне"
                >Окно</button>
            </div>
            {!isInlineEditingStatement ? (
                <>
                    <div style={{marginBottom: '10px'}}>
                        <strong style={labelStyle}>Краткое имя
(метка узла):</strong>
                        <p style={{ margin: 0, padding: '10px',
background: LIGHT_BACKGROUND, borderRadius:'0px',
border:`${BORDER_WIDTH} solid ${FAT_BORDER_COLOR}`, fontFamily:
FONT_FAMILY_TEXT }}>{stmtShortName || '(не задано)'}</p>
                    </div>
                    <div style={{marginBottom: '15px'}}>
                        <strong style={labelStyle}>Полный текст
(отображается в симуляторе):</strong>
                        <p style={{ margin: 0, padding: '10px',
background: LIGHT_BACKGROUND, borderRadius:'0px',
border:`${BORDER_WIDTH} solid ${FAT_BORDER_COLOR}`, whiteSpace:
'pre-wrap', maxHeight: '120px', overflowY: 'auto', fontFamily:
FONT_FAMILY_TEXT }}>{stmtFullText || '(не задан)'}</p>
                    </div>
                    <button onClick={() =>
setIsInlineEditingStatement(true)}
                        style={btnPrimaryStyles.base}
                        onMouseEnter={e =>
Object.assign(e.currentTarget.style, btnPrimaryStyles.hover)}
                        onMouseLeave={e =>
Object.assign(e.currentTarget.style, btnPrimaryStyles.base)}
                        onMouseDown={e =>
Object.assign(e.currentTarget.style, btnPrimaryStyles.active)}
                        onMouseUp={e =>
Object.assign(e.currentTarget.style, btnPrimaryStyles.hover)}
                    >Редактировать</button>
                </>
            ) : (
                <>
                  <div>
```

```
                    <label htmlFor="stmtShortName"
style={labelStyle}>Краткое имя (метка узла, макс 70):</label>
                    <input type="text" id="stmtShortName"
value={stmtShortName}
                      onChange={handleStmtShortNameChange}
                      maxLength={70}
                      style={inputBaseStyle} autoFocus
                      onFocus={e => Object.assign(e.target.style,
inputFocusStyle)}
                      onBlur={e => Object.assign(e.target.style,
inputBaseStyle)}/>
                  </div>
                  <div>
                    <label htmlFor="stmtFullText"
style={labelStyle}>Полный текст (макс 2000):</label>
                    <textarea id="stmtFullText"
value={stmtFullText}
                      onChange={handleStmtFullTextChange}
                      maxLength={2000}
                      style={textareaStyle}
                      onFocus={e => Object.assign(e.target.style,
inputFocusStyle)}
                      onBlur={e => Object.assign(e.target.style,
textareaStyle)}/>
              </div>
                  <div style={{ display: 'flex', gap: '10px',
marginTop: '10px' }}>
                    <button onClick={handleSaveStatementInline}
                        style={btnSuccessStyles.base}
                        onMouseEnter={e =>
Object.assign(e.currentTarget.style, btnSuccessStyles.hover)}
                        onMouseLeave={e =>
Object.assign(e.currentTarget.style, btnSuccessStyles.base)}
                        onMouseDown={e =>
Object.assign(e.currentTarget.style, btnSuccessStyles.active)}
                        onMouseUp={e =>
Object.assign(e.currentTarget.style, btnSuccessStyles.hover)}
                    >Сохранить</button>
                    <button onClick={handleCancelStatementInline}
                        style={btnDefaultStyles.base}
                        onMouseEnter={e =>
Object.assign(e.currentTarget.style, btnDefaultStyles.hover)}
                        onMouseLeave={e =>
Object.assign(e.currentTarget.style, btnDefaultStyles.base)}
                        onMouseDown={e =>
Object.assign(e.currentTarget.style, btnDefaultStyles.active)}
                        onMouseUp={e =>
Object.assign(e.currentTarget.style, btnDefaultStyles.hover)}
                    >Отмена</button>
                </div>
              </>
            )}
            </div>
```

```
        )}
        {node.type === 'statement' && (
          <div style={sectionStyle}>
            <h4 style={headingStyle(4)}>Реакции (ответы)</h4>
            <button onClick={() => { onAddOption(node.id);
handleAnyChange(); }}
                style={getButtonDynamicStyles('primary',
true).base}
                onMouseEnter={e =>
Object.assign(e.currentTarget.style,
getButtonDynamicStyles('primary', true).hover)}
                onMouseLeave={e =>
Object.assign(e.currentTarget.style,
getButtonDynamicStyles('primary', true).base)}
                onMouseDown={e =>
Object.assign(e.currentTarget.style,
getButtonDynamicStyles('primary', true).active)}
                onMouseUp={e =>
Object.assign(e.currentTarget.style,
getButtonDynamicStyles('primary', true).hover)}
            >Добавить реакцию</button>
            <div style={{marginTop: '15px'}}>
            {(node.data.options && node.data.options.length > 0) ?
(
              node.data.options.map((option) => (
                <OptionEditor
                  key={option.id} option={option}
optionData={optionDataStorage[option.id]}
                  onUpdate={onUpdateOption}
                  updateOptionInStorage={updateOptionInStorage}
                  onDelete={() => { if (window.confirm(`Удалить
реакцию?`)) onOptionDelete(node.id, option.id); }}
                  selectedOptionId={selectedOptionId}
setSelectedOptionId={setSelectedOptionId}
                  styles={{ labelStyle, inputBaseStyle,
textareaStyle, getButtonDynamicStyles, inputFocusStyle, hrStyle,
headingStyle, LIGHT_BACKGROUND, FAT_BORDER_COLOR, DARK_TEXT_COLOR,
ACCENT_COLOR_PRIMARY, ACCENT_COLOR_SUCCESS, ACCENT_COLOR_DANGER,
FONT_FAMILY_TEXT, FONT_FAMILY_HEADINGS, BORDER_WIDTH,
BUTTON_SHADOW, BUTTON_TRANSFORM_INITIAL, BUTTON_SHADOW_HOVER,
BUTTON_TRANSFORM_HOVER, BUTTON_SHADOW_ACTIVE,
BUTTON_TRANSFORM_ACTIVE, INPUT_FOCUS_BORDER_COLOR }}
                  parentNodeId={node.id}
                  clearErrorForElement={clearErrorForElement}

isError={node.data.errorOptionIds?.includes(option.id)}
                />
              ))
            ) : ( <p style={{textAlign: 'center', color:
DARK_TEXT_COLOR, opacity: 0.7, fontSize: '1em', marginTop: '10px',
fontFamily: FONT_FAMILY_TEXT}}>Нет реакций.</p> )}
            </div>
            </div>
```

```
          )}
      </div>
      <div style={editorFooterStyle}>
        <button onClick={handleDeleteNode}
            style={getButtonDynamicStyles('danger', true).base}
            onMouseEnter={e => Object.assign(e.currentTarget.style,
getButtonDynamicStyles('danger', true).hover)}
            onMouseLeave={e => Object.assign(e.currentTarget.style,
getButtonDynamicStyles('danger', true).base)}
            onMouseDown={e => Object.assign(e.currentTarget.style,
getButtonDynamicStyles('danger', true).active)}
            onMouseUp={e => Object.assign(e.currentTarget.style,
getButtonDynamicStyles('danger', true).hover)}
        >Удалить узел</button>
      </div>
      <Modal isOpen={statementModalIsOpen}
onRequestClose={handleCloseStatementModal}
contentLabel="Редактирование данных высказывания"
        style={{
          content: {
            width: 'clamp(450px, 65%, 850px)', top: '50%', left:
'50%', right: 'auto', bottom: 'auto',
            marginRight: '-50%', transform: 'translate(-50%, -
50%)',
            borderRadius: '0px', padding: '25px',
            background: LIGHT_BACKGROUND, border: `${BORDER_WIDTH}
solid ${FAT_BORDER_COLOR}`,
            boxShadow: `5px 5px 0px ${FAT_BORDER_COLOR}`,
            maxHeight: '85vh', display: 'flex', flexDirection:
'column',
            fontFamily: FONT_FAMILY_TEXT, color: DARK_TEXT_COLOR
          },
          overlay: { backgroundColor: 'rgba(0, 0, 0, 0.75)',
zIndex: 1050 }
        }}>
        <h2 style={{ ...headingStyle(), marginTop:0, fontSize:
'1.6em', fontFamily: FONT_FAMILY_HEADINGS }}>Редактировать
высказывание</h2>
        <div style={{flexGrow:1, overflowY:'auto', paddingRight:
'10px', marginBottom: '20px'}}>
            <div>
              <label htmlFor="modalStmtShortName"
style={labelStyle}>Краткое имя (метка узла, макс 70):</label>
              <input type="text" id="modalStmtShortName"
value={stmtShortName}
                onChange={handleStmtShortNameChange} maxLength={70}
                style={inputBaseStyle} autoFocus
                onFocus={e => Object.assign(e.target.style,
inputFocusStyle)}
                onBlur={e => Object.assign(e.target.style,
inputBaseStyle)}/>
            </div>
            <div>
```

```
              <label htmlFor="modalStmtFullText"
style={labelStyle}>Полный текст (макс 2000):</label>
              <textarea id="modalStmtFullText" value={stmtFullText}
                onChange={handleStmtFullTextChange}
maxLength={2000}
                style={{...textareaStyle, minHeight: '200px',
maxHeight: '40vh'}}
                onFocus={e => Object.assign(e.target.style,
inputFocusStyle)}
                onBlur={e => Object.assign(e.target.style,
{...textareaStyle, minHeight: '200px', maxHeight: '40vh'})}/>
            </div>
        </div>
        <div style={{ display: 'flex', gap: '10px', justifyContent:
'flex-end', paddingTop: '15px', borderTop: `${BORDER_WIDTH} solid
${FAT_BORDER_COLOR}` }}>
          <button onClick={handleSaveStatementModal}
            style={btnSuccessStyles.base}
            onMouseEnter={e => Object.assign(e.currentTarget.style,
btnSuccessStyles.hover)}
            onMouseLeave={e => Object.assign(e.currentTarget.style,
btnSuccessStyles.base)}
            onMouseDown={e => Object.assign(e.currentTarget.style,
btnSuccessStyles.active)}
            onMouseUp={e => Object.assign(e.currentTarget.style,
btnSuccessStyles.hover)}
            >Сохранить</button>
          <button onClick={handleCloseStatementModal}
            style={btnDefaultStyles.base}
            onMouseEnter={e => Object.assign(e.currentTarget.style,
btnDefaultStyles.hover)}
            onMouseLeave={e => Object.assign(e.currentTarget.style,
btnDefaultStyles.base)}
            onMouseDown={e => Object.assign(e.currentTarget.style,
btnDefaultStyles.active)}
            onMouseUp={e => Object.assign(e.currentTarget.style,
btnDefaultStyles.hover)}
            >Отмена</button>
          </div>
      </Modal>
    </div>
  );
};
const OptionEditor = ({
    option, optionData,
    onUpdate,
    updateOptionInStorage,
    onDelete,
    selectedOptionId, setSelectedOptionId, styles,
    parentNodeId,
    clearErrorForElement,
    isError,
}) => {
```

```
  const [isEditingThisOption, setIsEditingThisOption] =
useState(false);
  const [optionTextForHandle, setOptionTextForHandle] =
useState((option.text || "R").substring(0, 10));
  const [optDialogueText, setOptDialogueText] =
useState((optionData?.dialogueText || "").substring(0, 800));
  const [modalIsOpen, setModalIsOpen] = useState(false);
  const btnOptSave = styles.getButtonDynamicStyles('success');
  const btnOptModal = styles.getButtonDynamicStyles('default');
  const btnOptDelete = styles.getButtonDynamicStyles('danger');
  const btnOptCancel = styles.getButtonDynamicStyles('default',
true);
  const handleOptionAnyChange = () => {
    if (isError) {
        clearErrorForElement(option.id, 'option');
    }
  };
  useEffect(() => {
    setIsEditingThisOption(selectedOptionId === option.id);
  }, [selectedOptionId, option.id]);
  useEffect(() => {
    setOptionTextForHandle((option.text || `R`).substring(0, 10));
    setOptDialogueText((optionData?.dialogueText ||
"").substring(0, 800));
  }, [option, optionData]);
  const handleOptionTextForHandleChange = (e) => {
    setOptionTextForHandle(e.target.value.substring(0, 10));
    handleOptionAnyChange();
  };
  const handleOptDialogueTextChange = (e) => {
    setOptDialogueText(e.target.value.substring(0, 800));
    handleOptionAnyChange();
  };
  const handleSaveInline = () => {
    const finalHandleText = optionTextForHandle.substring(0, 10);
    const finalDialogueText = optDialogueText.substring(0, 800);
    updateOptionInStorage(option.id, finalHandleText,
finalDialogueText);
    onUpdate(parentNodeId, option.id, finalHandleText,
option.target);
    setIsEditingThisOption(false);
  };
  const handleCancelInline = () => {
    setOptionTextForHandle((option.text || `R`).substring(0, 10));
    setOptDialogueText((optionData?.dialogueText ||
"").substring(0, 800));
    setIsEditingThisOption(false);
    setSelectedOptionId(null);
  };
  const handleOpenModal = () => {
    setOptionTextForHandle((option.text || `R`).substring(0, 10));
    setOptDialogueText((optionData?.dialogueText ||
"").substring(0, 800));
```

```
    setModalIsOpen(true);
  };
  const handleCloseModal = () => setModalIsOpen(false);
  const handleSaveModal = () => {
    const finalHandleText = optionTextForHandle.substring(0, 10);
    const finalDialogueText = optDialogueText.substring(0, 800);
    updateOptionInStorage(option.id, finalHandleText,
finalDialogueText);
    onUpdate(parentNodeId, option.id, finalHandleText,
option.target);
    setModalIsOpen(false);
    setIsEditingThisOption(false);
    setSelectedOptionId(null);
  };
  const optionContainerStyle = {
    border: `${styles.BORDER_WIDTH} solid ${isError ?
styles.ACCENT_COLOR_DANGER : (isEditingThisOption ?
styles.ACCENT_COLOR_PRIMARY : styles.FAT_BORDER_COLOR)}`,
    background: isError ? '#ffebee' : (isEditingThisOption ?
styles.LIGHT_BACKGROUND : '#FFFFFF'),
    borderRadius: '0px',
    marginBottom: '10px',
    boxShadow: isEditingThisOption ? `2px 2px 0px
${styles.ACCENT_COLOR_PRIMARY}` : (isError ? `2px 2px 0px
${styles.ACCENT_COLOR_DANGER}` : styles.BUTTON_SHADOW),
    transition: 'border-color 0.1s, box-shadow 0.1s, background-
color 0.1s',
  };
  const optionHeaderStyle = {
    padding: '10px 12px',
    cursor: 'pointer',
    display: 'flex',
    justifyContent: 'space-between',
    alignItems: 'center',
    borderBottom: isEditingThisOption ? `${styles.BORDER_WIDTH}
solid ${isError ? styles.ACCENT_COLOR_DANGER :
styles.FAT_BORDER_COLOR}` : 'none',
    transition: 'background-color 0.1s, color 0.1s',
    background: isEditingThisOption ? styles.ACCENT_COLOR_PRIMARY :
'transparent',
    color: isEditingThisOption ? '#FFFFFF' :
styles.DARK_TEXT_COLOR,
    fontFamily: styles.FONT_FAMILY_HEADINGS,
  };
  const optionFormStyle = {
    padding: '12px',
    borderTop: isEditingThisOption ? `${styles.BORDER_WIDTH} solid
${styles.FAT_BORDER_COLOR}` : 'none',
    background: styles.LIGHT_BACKGROUND,
  };
  return (
    <div style={optionContainerStyle}>
      <div style={optionHeaderStyle}
```

```
        onClick={() => setSelectedOptionId(isEditingThisOption ?
null : option.id)}
        onMouseEnter={e => { if (!isEditingThisOption) {
e.currentTarget.style.background = styles.DARK_TEXT_COLOR;
e.currentTarget.style.color = styles.LIGHT_BACKGROUND; } }}
        onMouseLeave={e => { if (!isEditingThisOption) {
e.currentTarget.style.background = 'transparent';
e.currentTarget.style.color = styles.DARK_TEXT_COLOR; } }}
      >
        <span style={{ fontWeight: 'bold', color: isError ?
styles.ACCENT_COLOR_DANGER : 'inherit', fontSize: '1em' }}>
          {optionTextForHandle || '(пусто)'} {isError && <span
style={{ fontWeight: 'bold' }}> (ОШИБКА)</span>}
        </span>
        <span style={{ fontSize: '0.8em', color:
isEditingThisOption ? '#FFFFFF' : '#777', opacity:
isEditingThisOption ? 1 : 0.7, fontFamily: styles.FONT_FAMILY_TEXT
}}>ID: {option.id.substring(0,8)}</span>
      </div>
      {isEditingThisOption && (
        <div style={optionFormStyle}>
          <div style={{marginBottom: '10px'}}>
            <label htmlFor={`optHandleText-${option.id}`}
style={styles.labelStyle}>Текст на узле (макс 10 симв.):</label>
            <input type="text" id={`optHandleText-${option.id}`}
value={optionTextForHandle}
              onChange={handleOptionTextForHandleChange}
              maxLength={10}
              style={styles.inputBaseStyle} autoFocus
              onFocus={e => Object.assign(e.target.style,
styles.inputFocusStyle)}
              onBlur={e => Object.assign(e.target.style,
styles.inputBaseStyle)}/>
          </div>
          <div style={{marginBottom: '10px'}}>
            <label htmlFor={`optDialogueText-${option.id}`}
style={styles.labelStyle}>Текст реакции игрока (макс 800
симв.):</label>
            <textarea id={`optDialogueText-${option.id}`}
value={optDialogueText}
              onChange={handleOptDialogueTextChange}
              maxLength={800}
              style={{...styles.inputBaseStyle, minHeight: '100px',
resize: 'vertical', lineHeight: 1.5}}
              onFocus={e => Object.assign(e.target.style,
styles.inputFocusStyle)}
              onBlur={e => Object.assign(e.target.style,
{...styles.inputBaseStyle, minHeight: '100px', resize: 'vertical',
lineHeight: 1.5} )}/>
          </div>
          <div style={{ display: 'flex', gap: '8px', marginTop:
'15px', alignItems: 'center' }}>
            <button onClick={handleSaveInline}
```

```
                style={btnOptSave.base}
                onMouseEnter={e =>
Object.assign(e.currentTarget.style, btnOptSave.hover)}
                onMouseLeave={e =>
Object.assign(e.currentTarget.style, btnOptSave.base)}
                onMouseDown={e =>
Object.assign(e.currentTarget.style, btnOptSave.active)}
                onMouseUp={e =>
Object.assign(e.currentTarget.style, btnOptSave.hover)}
            >Сохранить</button>
            <button onClick={handleOpenModal}
                style={{...btnOptModal.base, padding: '8px 10px',
fontSize: '0.9em'}}
                onMouseEnter={e =>
Object.assign(e.currentTarget.style, btnOptModal.hover)}
                onMouseLeave={e =>
Object.assign(e.currentTarget.style, btnOptModal.base)}
                onMouseDown={e =>
Object.assign(e.currentTarget.style, btnOptModal.active)}
                onMouseUp={e =>
Object.assign(e.currentTarget.style, btnOptModal.hover)}
            >Окно</button>
            <button onClick={onDelete}
                style={{ ...btnOptDelete.base, padding: '8px 10px',
fontSize: '0.9em', marginLeft: 'auto'}}
                onMouseEnter={e =>
Object.assign(e.currentTarget.style, btnOptDelete.hover)}
                onMouseLeave={e =>
Object.assign(e.currentTarget.style, btnOptDelete.base)}
                onMouseDown={e =>
Object.assign(e.currentTarget.style, btnOptDelete.active)}
                onMouseUp={e =>
Object.assign(e.currentTarget.style, btnOptDelete.hover)}
             title="Удалить реакцию">✕</button>
          </div>
           <button onClick={handleCancelInline}
                style={{...btnOptCancel.base, marginTop: '10px'}}
                onMouseEnter={e =>
Object.assign(e.currentTarget.style, btnOptCancel.hover)}
                onMouseLeave={e =>
Object.assign(e.currentTarget.style, btnOptCancel.base)}
                onMouseDown={e =>
Object.assign(e.currentTarget.style, btnOptCancel.active)}
                onMouseUp={e =>
Object.assign(e.currentTarget.style, btnOptCancel.hover)}
            >Отменить редактирование</button>
        </div>
      )}
      <Modal isOpen={modalIsOpen} onRequestClose={handleCloseModal}
contentLabel="Редактирование реакции"
        style={{
          content: {
```

```
            width: 'clamp(400px, 50%, 700px)', top: '50%', left:
'50%', right: 'auto', bottom: 'auto',
            marginRight: '-50%', transform: 'translate(-50%, -
50%)',
            borderRadius: '0px', padding: '25px',
            background: styles.LIGHT_BACKGROUND, border:
`${styles.BORDER_WIDTH} solid ${styles.FAT_BORDER_COLOR}`,
            boxShadow: `5px 5px 0px ${styles.FAT_BORDER_COLOR}`,
            maxHeight: '80vh', display: 'flex', flexDirection:
'column',
            fontFamily: styles.FONT_FAMILY_TEXT, color:
styles.DARK_TEXT_COLOR
          },
          overlay: { backgroundColor: 'rgba(0, 0, 0, 0.75)',
zIndex: 1050 }
        }}>
        <h2 style={{ ...styles.headingStyle(), marginTop:0,
fontSize: '1.3em', fontFamily: styles.FONT_FAMILY_HEADINGS
}}>Редактировать реакцию</h2>
        <div style={{flexGrow:1, overflowY:'auto', paddingRight:
'10px', marginBottom: '20px'}}>
            <div style={{marginBottom: '10px'}}>
                <label htmlFor={`modalOptHandleText-${option.id}`}
style={styles.labelStyle}>Текст на узле (макс 10 симв.):</label>
                <input type="text" id={`modalOptHandleText-
${option.id}`} value={optionTextForHandle}
                    onChange={handleOptionTextForHandleChange}
maxLength={10}
                    style={styles.inputBaseStyle} autoFocus
                    onFocus={e => Object.assign(e.target.style,
styles.inputFocusStyle)}
                    onBlur={e => Object.assign(e.target.style,
styles.inputBaseStyle)}/>
            </div>
            <div style={{marginBottom: '10px'}}>
                <label htmlFor={`modalOptDialogueText-
${option.id}`} style={styles.labelStyle}>Текст реакции игрока (макс
800 симв.):</label>
                <textarea id={`modalOptDialogueText-${option.id}`}
value={optDialogueText}
                    onChange={handleOptDialogueTextChange}
maxLength={800}
                    style={{...styles.inputBaseStyle, minHeight:
'150px', maxHeight: '35vh', resize: 'vertical', lineHeight: 1.5}}
                    onFocus={e => Object.assign(e.target.style,
styles.inputFocusStyle)}
                    onBlur={e => Object.assign(e.target.style,
{...styles.inputBaseStyle, minHeight: '150px', maxHeight: '35vh',
resize: 'vertical', lineHeight: 1.5} )}/>
            </div>
        </div>
```

```
        <div style={{ display: 'flex', gap: '10px', justifyContent:
'flex-end', paddingTop: '15px', borderTop: `${styles.BORDER_WIDTH}
solid ${styles.FAT_BORDER_COLOR}` }}>
          <button onClick={handleSaveModal}
            style={styles.getButtonDynamicStyles('success').base}
            onMouseEnter={e => Object.assign(e.currentTarget.style,
styles.getButtonDynamicStyles('success').hover)}
            onMouseLeave={e => Object.assign(e.currentTarget.style,
styles.getButtonDynamicStyles('success').base)}
            onMouseDown={e => Object.assign(e.currentTarget.style,
styles.getButtonDynamicStyles('success').active)}
            onMouseUp={e => Object.assign(e.currentTarget.style,
styles.getButtonDynamicStyles('success').hover)}
            >Сохранить и закрыть</button>
          <button onClick={handleCloseModal}
            style={styles.getButtonDynamicStyles('default').base}
            onMouseEnter={e => Object.assign(e.currentTarget.style,
styles.getButtonDynamicStyles('default').hover)}
            onMouseLeave={e => Object.assign(e.currentTarget.style,
styles.getButtonDynamicStyles('default').base)}
            onMouseDown={e => Object.assign(e.currentTarget.style,
styles.getButtonDynamicStyles('default').active)}
            onMouseUp={e => Object.assign(e.currentTarget.style,
styles.getButtonDynamicStyles('default').hover)}
            >Отмена</button>
        </div>
      </Modal>
    </div>
  );
};
export default NodeEditor;
```

Приложение В

(рекомендуемое)

Листинг автоматизированных тестов

```
import React from 'react';
import { render, screen, fireEvent, act } from '@testing-
library/react';
import '@testing-library/jest-dom';
import MainMenu from '../views/MainMenu';
import MetaEditorSidebar from '../views/MetaEditorSidebar';
import ProjectViewSidebar from '../views/ProjectViewSidebar';
import DialogSimulatorModal from
'../components/DialogSimulatorModal';
import ProjectView from '../views/ProjectView';
import { StartNode } from '../components/StartNode';
import DialogReferenceNode from
'../components/DialogReferenceNode';
import DialogIconNode from '../components/DialogIconNode';

jest.mock('react-modal', () => {
    const Modal = ({ children, isOpen, contentLabel,
onRequestClose, style }) => {
        if (!isOpen) return null;
        return (
            <div data-testid="mock-modal" aria-
label={contentLabel} style={style?.content}>
                {children}
                <button onClick={onRequestClose}>Close Mock
Modal</button>
            </div>
        );
    };
    Modal.setAppElement = jest.fn();
    return Modal;
});

jest.mock('html-to-image', () => ({
    toPng: jest.fn().mockResolvedValue('fake_png_data_url'),
    toJpeg: jest.fn().mockResolvedValue('fake_jpeg_data_url'),
}));

jest.mock('uuid', () => ({
    v4: jest.fn(() => 'mock-uuid-1234'),
}));

jest.mock('react-flow-renderer', () => {
    const originalModule = jest.requireActual('react-flow-
renderer');
    return {
        ...originalModule,
        useStore: jest.fn(() => ({ edges: [] })),
        Handle: ({ type, position, id, style, isConnectable,
children }) => (
            <div data-testid={`handle-${type}-${position}-${id ||
'default'}`} style={style}>
                {children}
            </div>
```

```
        ),
        Controls: () => <div data-testid="mock-controls" />,
        Background: () => <div data-testid="mock-background" />,
    };
});


describe('MainMenu Component', () => {
    const mockOnCreateNew = jest.fn();
    const mockOnOpenProject = jest.fn();

    beforeEach(() => {
        jest.clearAllMocks();
    });

    test('рендерит заголовок и кнопки', () => {
        render(<MainMenu onCreateNew={mockOnCreateNew}
onOpenProject={mockOnOpenProject} />);
        expect(screen.getByText(/Веб-инструмент для создания и
редактирования интерактивных диалогов/i)).toBeInTheDocument();
        expect(screen.getByText(/Создать новый
проект/i)).toBeInTheDocument();
        expect(screen.getByText(/Открыть
проект/i)).toBeInTheDocument();
    });
    test('вызывает onCreateNew при нажатии на кнопку "Создать
новый проект"', () => {
        render(<MainMenu onCreateNew={mockOnCreateNew}
onOpenProject={mockOnOpenProject} />);
        fireEvent.click(screen.getByText(/Создать новый
проект/i));
        expect(mockOnCreateNew).toHaveBeenCalledTimes(1);
    });

    test('при нажатии "Открыть проект" кликает по скрытому input
file', () => {
        render(<MainMenu onCreateNew={mockOnCreateNew}
onOpenProject={mockOnOpenProject} />);
        const openProjectButton = screen.getByText(/Открыть
проект/i);
        const fileInput =
document.getElementById('openProjectFile');
        const clickSpy = jest.spyOn(fileInput,
'click').mockImplementation(() => {});

        fireEvent.click(openProjectButton);
        expect(clickSpy).toHaveBeenCalledTimes(1);
        clickSpy.mockRestore();
    });

    test('вызывает onOpenProject при выборе файла', () => {
        render(<MainMenu onCreateNew={mockOnCreateNew}
onOpenProject={mockOnOpenProject} />);
```

```
        const fileInput =
document.getElementById('openProjectFile');
        const mockFile = new File(['dummy content'],
'project.dialogproject.json', { type: 'application/json' });

        fireEvent.change(fileInput, { target: { files: [mockFile]
} });
        expect(mockOnOpenProject).toHaveBeenCalledWith(mockFile);
    });
});

describe('MetaEditorSidebar Component', () => {
    const mockOnDragStart = jest.fn();
    const dialogs = [
        { id: '1', name: 'Dialog 1', isComplete: true,
entryPoints: [], exitPoints: [] },
        { id: '2', name: 'Dialog 2', isComplete: false,
entryPoints: [], exitPoints: [] },
    ];

    test('рендерит заголовок и диалоги', () => {
        render(<MetaEditorSidebar dialogs={dialogs}
onDragStart={mockOnDragStart} />);
        expect(screen.getByText(/Диалоги
\(Перетащите\)/i)).toBeInTheDocument();
        expect(screen.getByText(/Dialog 1/)).toBeInTheDocument();
        expect(screen.getByText(/Dialog 2/)).toBeInTheDocument();
    });

    test('завершенные диалоги можно перетаскивать', () => {
        render(<MetaEditorSidebar dialogs={dialogs}
onDragStart={mockOnDragStart} />);
        const dialog1 = screen.getByText(/Dialog
1/).closest('div');
        expect(dialog1).toHaveAttribute('draggable', 'true');
    });

    test('незавершенные диалоги нельзя перетаскивать', () => {
        render(<MetaEditorSidebar dialogs={dialogs}
onDragStart={mockOnDragStart} />);
        const dialog2 = screen.getByText(/Dialog
2/).closest('div');
        expect(dialog2).toHaveAttribute('draggable', 'false');
    });



    test('onDragStart вызывается для завершенного диалога', () =>
{
        render(<MetaEditorSidebar dialogs={dialogs}
onDragStart={mockOnDragStart} />);
        const dialog1Div = screen.getByText(/Dialog
1/).closest('div');
```

```
        const mockDataTransfer = { setData: jest.fn(),
effectAllowed: '' };

        fireEvent.dragStart(dialog1Div, { dataTransfer:
mockDataTransfer });

        expect(mockDataTransfer.setData).toHaveBeenCalledWith(
            'application/reactflow',
            JSON.stringify({
                type: 'dialogRef',
                dialogId: '1',
                label: 'Dialog 1',
                entryPoints: [],
                exitPoints: []
            })
        );
        expect(mockOnDragStart).toHaveBeenCalledTimes(1);
    });
});

describe('ProjectViewSidebar Component', () => {
    const mockAdd = jest.fn();
    const mockImport = jest.fn();
    const mockSave = jest.fn();
    const mockClose = jest.fn();
    const mockOpenMeta = jest.fn();

    test('рендерит все кнопки', () => {
        render(<ProjectViewSidebar
            onAddDialog={mockAdd}
onImportDialogFromFile={mockImport} onSaveProject={mockSave}
            onCloseProject={mockClose}
onOpenMetaEditor={mockOpenMeta}
        />);
        expect(screen.getByText(/Добавить новый
диалог/i)).toBeInTheDocument();
        expect(screen.getByText(/Импорт диалога из
файла/i)).toBeInTheDocument();
        expect(screen.getByText(/Открыть Мета-
Редактор/i)).toBeInTheDocument();
        expect(screen.getByText(/Сохранить весь
проект/i)).toBeInTheDocument();
        expect(screen.getByText(/Закрыть
проект/i)).toBeInTheDocument();
    });

    test('вызывает колбэки при кликах', () => {
        render(<ProjectViewSidebar
            onAddDialog={mockAdd}
onImportDialogFromFile={mockImport} onSaveProject={mockSave}
            onCloseProject={mockClose}
onOpenMetaEditor={mockOpenMeta}
        />);
```

```
        fireEvent.click(screen.getByText(/Добавить новый
диалог/i));
        expect(mockAdd).toHaveBeenCalledTimes(1);

        const importButton = screen.getByText(/Импорт диалога из
файла/i);
        const fileInput =
document.getElementById('importDialogFileAsNew');
        const clickSpy = jest.spyOn(fileInput,
'click').mockImplementation(() => {});
        fireEvent.click(importButton);
        expect(clickSpy).toHaveBeenCalledTimes(1);
        clickSpy.mockRestore();

        fireEvent.click(screen.getByText(/Открыть Мета-
Редактор/i));
        expect(mockOpenMeta).toHaveBeenCalledTimes(1);
        fireEvent.click(screen.getByText(/Сохранить весь
проект/i));
        expect(mockSave).toHaveBeenCalledTimes(1);
        fireEvent.click(screen.getByText(/Закрыть проект/i));
        expect(mockClose).toHaveBeenCalledTimes(1);
    });
});


describe('Node Components', () => {
    const baseProps = { id: 'node-1', selected: false,
isConnectable: true, dragging: false, xPos: 0, yPos: 0, zIndex:1
};

    test('StartNode.Component рендерит метку', () => {
        render(<StartNode.Component {...baseProps} data={{ label:
'Старт Тест' }} />);
        expect(screen.getByText('Старт
Тест')).toBeInTheDocument();
    });


    test('DialogReferenceNode рендерит метку и ID диалога', () =>
{
        const dialogRefData = { label: 'Ссылка на диалог',
dialogId: 'dialog-abc', entryPoints: [], exitPoints: [] };
        render(<DialogReferenceNode {...baseProps}
data={dialogRefData} />);
        expect(screen.getByText('Ссылка на
диалог')).toBeInTheDocument();
        expect(screen.getByText(/ID: dialog-
a/i)).toBeInTheDocument();
    });

    test('DialogIconNode рендерит метку и статус', () => {
        render(<DialogIconNode {...baseProps} data={{ label:
```

```
'Иконка Диалога', isComplete: true }} onDoubleClick={jest.fn()}
/>);
        expect(screen.getByText('Иконка
Диалога')).toBeInTheDocument();
        expect(screen.getByText('Завершен')).toBeInTheDocument();
    });
});

describe('DialogSimulatorModal Component', () => {
    const mockOnRequestClose = jest.fn();

    test('подгружает элементы диалога', () => {
        render(<DialogSimulatorModal
            isOpen={true}
            onRequestClose={mockOnRequestClose}
            nodes={[]} edges={[]} statementData={{}}
optionData={{}}
        />);
        const modalContent = screen.getByTestId('mock-modal');
        expect(modalContent).toHaveTextContent(/Симулятор/i);
    });


});

describe('ProjectView Component', () => {
    const mockProjectData = {
        name: 'Мой Проект',
        dialogs: [],
        metaGraph: { nodes: [], edges: [] },
    };
    const mockUpdate = jest.fn();
    const mockOpenDialog = jest.fn();
    const mockOpenMeta = jest.fn();
    const mockClose = jest.fn();

    test('рендерит информацию о проекте', () => {
        render(<ProjectView
            projectData={mockProjectData}
            updateProjectData={mockUpdate}
            onOpenDialog={mockOpenDialog}
            onOpenMetaEditor={mockOpenMeta}
            onCloseProject={mockClose}
        />);
        expect(screen.getByText(`Проект:
${mockProjectData.name}`)).toBeInTheDocument();
        expect(screen.getByText(/В проекте нет
диалогов/i)).toBeInTheDocument();
        expect(screen.getByPlaceholderText(/Поиск
диалогов/i)).toBeInTheDocument();
    });
});
```