МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий
Кафедра компьютерных технологий

Направление подготовки 09.03.01 Информатика и вычислительная техника
Направленность (профиль): Программная инженерия и компьютерные науки

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Ли Константина Алексеевича

Тема работы:

РАЗРАБОТКА МОДУЛЯ ВИЗУАЛИЗАЦИИ ДЛЯ ОБЛАЧНОГО ПРАКТИКУМА ПО ЯЗЫКУ POST

«К защите допущена»
Заведующий кафедрой,
д.т.н., доцент
Зюбин В.Е. /……..
(ФИО) / (подпись)
« » мая 2024 г.

Руководитель ВКР
д.т.н., доцент
зав. каф. КТ ФИТ НГУ
Зюбин В.Е. /……...
(ФИО) / (подпись)
« » мая 2024 г.

Соруководитель ВКР
к.ф-м.н., доцент каф. СИ ФИТ НГУ
Ануреев И.С. /……...
(ФИО) / (подпись)
« » мая 2024 г.

Новосибирск, 2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)
Факультет информационных технологий
Кафедра компьютерных технологий
(название кафедры)
Направление подготовки 09.03.01 Информатика и вычислительная техника
Направленность (профиль): Программная инженерия и компьютерные науки

УТВЕРЖДАЮ

Зав. кафедрой Зюбин В.Е.
(фамилия, И., О.)
…….
(подпись)
«……»……20…г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

Студенту Ли Константину Алексеевичу, группы 20207
(фамилия, имя, отчество, номер группы)

Тема «Разработка модуля визуализации для облачного практикума по языку poST»
(полное название темы выпускной квалификационной работы)

утверждена распоряжением проректора по учебной работе от…….№……

Срок сдачи студентом готовой работы……2024 г.

Исходные данные (или цель работы): Создание модулей динамического отображения, редактирования и экспорт для облачного практикума по языку poST.

Структурные части работы: Анализ и концептуализация графического редактора, обзор существующих технологий и инструментов, формулировка требований к модулю, проектирование архитектуры, реализация и апробация.

Срок сдачи студентом готовой работы……2024 г.

Исходные данные (или цель работы): Создание модулей динамического отображения, редактирования и экспорт для облачного практикума по языку poST.

Консультанты по разделам ВКР (при необходимости, с указанием разделов):

_____
(раздел, ФИО)

Руководитель ВКР
д.т.н., доцент, зав. каф. компьютерных технологий ФИТ
Зюбин В.Е. /……...
(ФИО) / (подпись)
«…»……20…г.

Соруководитель ВКР
к.ф-м.н., доцент каф. СИ ФИТ НГУ
Ануреев И.С. /……...
(ФИО) / (подпись)
«…»……20…г.

Задание принял к исполнению
Ли К.А. /……...
(ФИО студента) / (подпись)
«…»……20…г.

Содержание

ВВЕДЕНИЕ

С развитием технологий и углублением интеграции цифровых решений во все аспекты жизни, облачные технологии становятся неотъемлемой частью образовательного процесса. Их применение в сфере обучения программированию и автоматизации открывает новые возможности для гибкости и масштабируемости учебных программ. Наглядное представление информации помогает студентам лучше усваивать сложные концепции и разрабатывать более эффективные решения в области программирования и автоматизации.

В современном образовательном процессе по программированию особое место занимает интерактивность и визуализация. Визуализация алгоритмов и процессов не только способствует более глубокому пониманию материала, но и позволяет студентам наблюдать за динамикой выполнения программ, что особенно важно в области автоматизации и управления. Применение интерактивных графических редакторов и инструментов визуализации в облачных платформах обучения может значительно повысить качество и эффективность образовательного процесса.

Модуль визуализации, включающий в себя модули динамического отображения, редактирования и экспорта будет обеспечивать создание, редактирование и визуализацию графических элементов, представляющих физические и управляющие компоненты системы, а также их динамическое взаимодействие в зависимости от изменений параметров в реальном времени.

Цель работы: создание модулей динамического отображения, редактирования и экспорт для облачного практикума по языку poST.

Для достижения этой цели поставлены следующие задачи:

- ➢ Анализ технологий виртуализации объекта управления и специфики существующего подхода.
- ➢ Составление требований.
- ➢ Определение языка спецификации визуального представления ОУ
- ➢ Реализация инструментальных средств

Научная новизна работы заключается в разработке комплекса визуализации состоящего из модулей редактирования, визуализации и модуля экспортирования для облачных практикумов по языку poST, который интегрируется с системой виртуализации управляющих алгоритмов. Отличительной особенностью разработанного редактора является возможность визуального проектирования моделей систем управления с последующей их интерактивной визуализацией и динамической связью с переменными программы управления. Это позволяет отображать изменения в реальном времени, повышая наглядность и понимание процессов управления.

Практическая ценность работы выражается в создании инструмента, который упрощает процесс обучения разработке ПО систем управления. Разработанный графический редактор позволяет преподавателям и студентам визуализировать и тестировать управляющие алгоритмы в интерактивной среде, способствуя пониманию работы систем управления и ускорению обучения.

Глава 1. Анализ и концептуализация графического редактора для систем управления

1.1 Особенности задач виртуализации объектов управления в промышленной автоматизации

Промышленная автоматизация включает в себя разработку и реализацию алгоритмов управления для различных производственных процессов и систем. Эти алгоритмы должны обеспечивать точное и надежное управление механическими и электронными устройствами на производстве. Однако разработка и тестирование таких алгоритмов на реальных объектах управления могут сопряжены с высокими рисками, включая возможность технических сбоев, финансовые потери и даже угрозы безопасности персонала и окружающей среды. Виртуализация объектов управления является одним из эффективных решений этих проблем, позволяя разработчикам создавать и тестировать алгоритмы в безопасной, контролируемой среде без воздействия на реальные процессы.

Виртуальные объекты управления (ВОУ) [1] — это программные модели, которые имитируют поведение и характеристики реальных систем и процессов. Это позволяет инженерам выполнять различные операции, включая:

- Моделирование и симуляция: Виртуальные объекты управления могут моделировать физические и логические процессы, которые участвуют в промышленной автоматизации, обеспечивая возможность анализа и оптимизации алгоритмов управления.
- Тестирование и отладка: ВОУ облегчают процесс отладки, позволяя разработчикам тестировать алгоритмы управления в различных сценариях и условиях, воспроизводить и исправлять ошибки без риска для реального оборудования.
- Обучение и демонстрация: ВОУ могут быть использованы для обучения операторов и технических специалистов, предоставляя им возможность

изучать работу комплексных систем управления в упрощенной и понятной форме.

Создание ВОУ требует применения специализированного программного обеспечения, которое может точно воспроизводить физические и логические процессы реальных объектов. Основные аспекты, на которые следует обратить внимание при разработке виртуальных объектов управления, включают:

- Точность моделирования: важно, чтобы модели точно отражали реальные процессы и характеристики объектов управления, чтобы результаты тестов были достоверными.
- Интерактивность и реалистичность: ВОУ должны предоставлять реалистичный и интерактивный интерфейс, позволяющий пользователю взаимодействовать с моделью так, как если бы это был реальный объект.
- Производительность: Системы должны обеспечивать высокую производительность и быстродействие даже при моделировании сложных и многокомпонентных процессов.

Таким образом, разработка и внедрение ВОУ в образовательные и исследовательские проекты способствует повышению безопасности, снижению затрат и ускорению процесса разработки алгоритмов управления. Это делает виртуализацию неотъемлемым инструментом в современной промышленной автоматизации.

1.2 Существующие технологии и инструменты для графического редактирования

Современный рынок предлагает множество технологий и инструментов для графического редактирования, которые применяются в разработке систем управления. Данный раздел представляет обзор ключевых технологий и инструментов, выделяя их основные характеристики и области применения.

1. *LabVIEW (Laboratory Virtual Instrument Engineering Workbench)* [2]: Платформа и среда разработки от National Instruments, широко используется для сбора и обработки данных, управления техническими объектами и технологическими процессами. LabVIEW основана на графическом языке программирования G и предлагает возможности для создания пользовательских интерфейсов, обработки сигналов и взаимодействия с аппаратным обеспечением.

2. *Simulink om MathWorks*: Графическая среда для моделирования, анализа и симуляции многодоменных динамических систем. Simulink широко используется в области управления, сигнальной обработки, системной инженерии и других технических дисциплинах. Предлагает интуитивно понятный интерфейс перетаскивания блоков для создания сложных моделей.

3. *CoDeSys (Controller Development System)* [3]: Интегрированная среда разработки для программирования промышленных контроллеров в соответствии со стандартом IEC 61131-3. CoDeSys поддерживает все основные языки программирования этого стандарта и обладает широкими возможностями для визуализации процессов.

4. *Factory I/O om Real Games*: Интерактивная среда (sandbox) для обучения автоматизации с использованием ПЛК. Предоставляет инструменты для создания 3D-сценариев промышленных процессов и их взаимодействия с ПЛК, поддерживая реалистичную симуляцию и визуализацию.

5. *ISaGRAF*: Графическая среда разработки для создания приложений для ПЛК на языках стандарта IEC 61131-3 и IEC 61499. Поддерживает создание распределенных систем управления и предлагает гибкие инструменты для визуализации и отладки.

Приведённые программные продукты демонстрируют эффективные способы визуализации и интерактивности, однако все они имеют общие ограничения, связанные с применением специализированных текстовых языков и синтаксических структур для описания визуальных компонентов, что увеличивает сложность разработки и требует специфических знаний в области графического

дизайна и программирования. Кроме того, лицензионные ограничения, проблемы с доступностью платежных средств, а также зависимость от определённых платформ и операционных систем существенно снижают гибкость использования существующих решений и увеличивают их стоимость, что делает их менее доступными для образовательных и исследовательских учреждений.

1.3 Формулировка требований к разрабатываемому модулю

На основании проведенного анализа предметной области и сформулированных требований к функциональности были сформулированы требования к реализации:

1. Визуализация

Модуль визуализации должен предоставлять комплексные инструменты для детальной настройки визуальных элементов, что включает возможность манипулирования параметрами таких элементов, как размеры, цвета, текстуры и шрифты. Должна быть реализована функциональность, позволяющая пользователям настраивать интерфейс в соответствии с конкретными задачами и предпочтениями. Наряду с возможностями настройки, редактор должен включать библиотеку предопределённых элементов управления, таких как индикаторы состояния, регуляторы и переключатели, что позволит облегчить процесс создания интерфейсов для мониторинга и управления. Эти элементы должны быть легко интегрируемы и настраиваемы в соответствии с различными сценариями использования.

2. Совместимость

Система должна обеспечивать высокую степень совместимости с внешними системами управления и другими инструментами анализа данных. Для этого необходимо предусмотреть поддержку стандартных протоколов данных и интерфейсов API, что позволит осуществлять обмен данными в режиме реального времени. Такое взаимодействие должно поддерживать как загрузку данных

из внешних источников, так и выгрузку обработанных данных и состояний системы для дальнейшего использования в других приложениях. Это требование важно для интеграции модуля в различные инфраструктуры и его использование в качестве части более крупных систем.

3. Динамическое отображение

Критически важной является способность редактора динамически отображать изменения в данных и состояниях системы. Редактор должен обладать функциональностью, позволяющей визуализировать эти изменения в режиме реального времени, что требует от системы быстрого реагирования на входные данные и их немедленного отображения. Интерфейс должен предусматривать взаимодействие пользователя с визуальными элементами, такими как управление параметрами через графический интерфейс, и обработку пользовательских команд, что требует от системы поддержки событийного программирования и обработки событий в пользовательском интерфейсе.

4. Функциональность экспорта

Модуль должен предоставлять возможности для экспорта визуализаций в различные форматы, сохраняя при этом всю встроенную логику поведения элементов, включая анимации и интерактивные функции. Это позволит пользователям сохранять и делиться своими работами, а также использовать созданные визуализации в других проектах или системах. Поддержка стандартных форматов данных, таких как JSON, XML и других, должна обеспечивать лёгкость интеграции и переносимости созданных интерфейсов.

5. Спецификация виртуального объекта управления

Модуль должен включать инструменты для визуально-блочного описания спецификаций виртуальных объектов управления, что позволит пользователям графически создавать и настраивать логику работы системы. Это должно включать возможность определения поведения элементов через визуальные блоки,

что сделает процесс программирования доступным не только для разработчиков, но и для специалистов, не имеющих глубоких знаний в кодировании.

6. Языковые средства

Выбранные для разработки языковые средства должны поддерживать спецификацию поведения визуальных элементов через события и временные интервалы, обеспечивая тем самым возможность создания динамически изменяющихся интерфейсов, которые могут адаптироваться к изменениям условий операционной среды. Это включает поддержку условных операторов, циклов и других структур управления, которые позволяют определять комплексные взаимодействия внутри системы.

В представленном анализе предметной области и определении функциональных требований к системе визуализации были идентифицированы ключевые аспекты. В заключение, можно конкретизировать сформулированные требования следующим образом:

1. **Инструменты визуализации**: Редактор должен включать инструменты для настройки элементов, а также библиотеку элементов управления, таких как индикаторы и регуляторы.

2. **Совместимость**: Редактор должен обеспечивать возможность интеграции и обмена данными с внешними системами управления.

3. **Динамическое отображение**: Среда исполнения должна отображать динамические изменения и обрабатывать пользовательские взаимодействия с элементами.

4. **Функциональность экспорта**: Должна быть предусмотрена возможность экспортировать визуализации с сохранением встроенной логики поведения элементов, включая анимацию.

5. **Спецификация виртуального ОУ**: Для спецификации ОУ должен использоваться визуально-блочный язык.

6.**Выбранные языковые средства**: Должна быть предусмотрена возможность задавать реакцию элементов графического интерфейса на события, в том числе, временные.

1.4 Архитектура и место в проекте

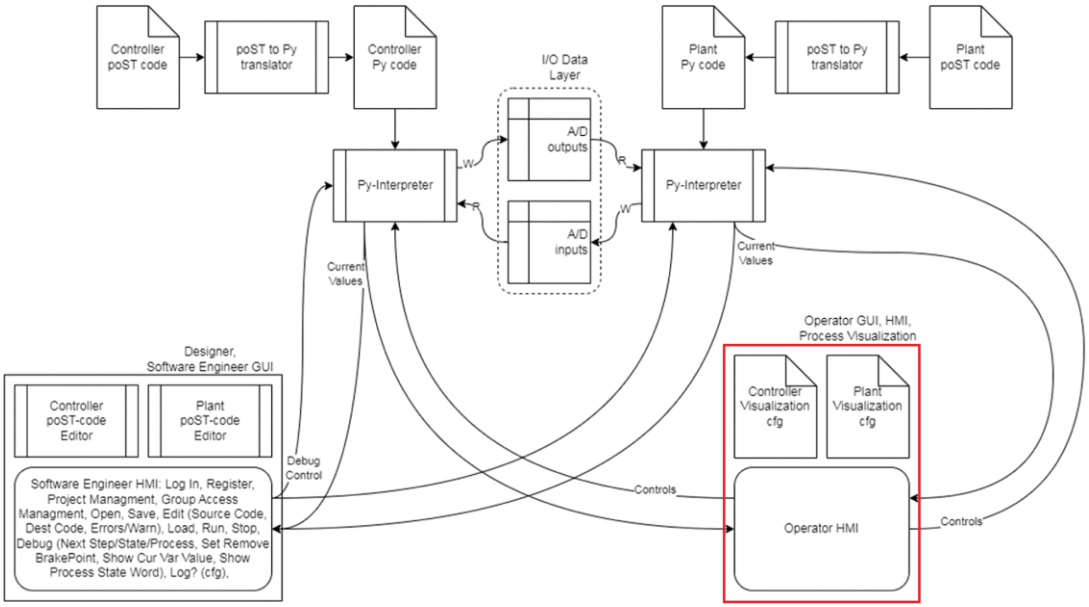

Рисунок 1. Архитектура проекта

Представленная схема описывает структуру системы, предназначенной для интеграции и взаимодействия различных компонентов программного и аппаратного обеспечения, в рамках управления и визуализации процессов виртуального объекта управления (ОУ) и контроллера.

Компоненты системы:

1. Controller poST Code и Plant poST Code:

Это исходные коды, написанные на языке poST для контроллера и объекта управления соответственно. Они служат основой для моделирования и управления процессами.

2. poST to Py Translator [4]:

Трансляторы, которые переводят код poST в Python. Это обеспечивает возможность исполнения логики управления с помощью Python-интерпретаторов, позволяя использовать широкий спектр библиотек и инструментов Python.

3. Py Interpreter:

Python-интерпретаторы обрабатывают транслированный код для контроллера и объекта управления. Они выполняют бизнес-логику и управление процессами, получая текущие значения процессов и отправляя команды управления.

4. I/O Data Layer с A/D Inputs и A/D Outputs:

Слой ввода/вывода данных, включающий аналого-цифровые (A/D) преобразователи для считывания сигналов из внешнего мира и отправки управляющих сигналов. Эти компоненты обеспечивают связь между физическими или виртуальными объектами и системой.

5. Operator GUI, HMI, Process Visualization [5]:

Компонента представляет из себя модуль визуализации, представленный в данной работе. Содержит в себе интерфейс оператора и «человеко-машинный интерфейс» (HMI), предоставляющие функциональность визуализации процессов. Модуль позволяет операторам конструировать визуальное представление ОУ в контексте кибер-физической системы, импортировать или экспортировать конфигурации различных кибер-физических системам, наблюдать за текущим состоянием процессов, а также управлять ими через визуальные элементы управления.

Глава 2. Проектирование

2.1 Архитектура модуля

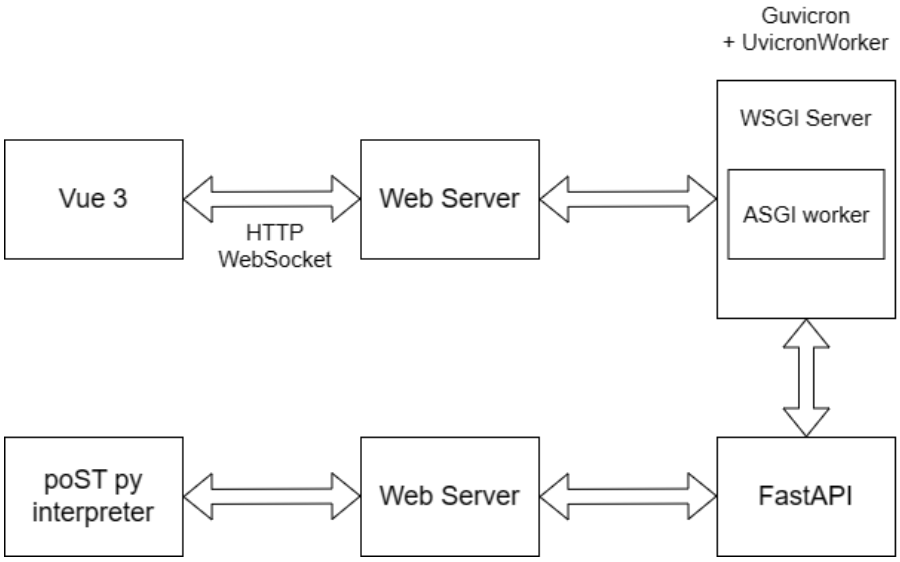

Рисунок 2. Архитектура модуля визуализации

- Vue 3 Frontend [6]:

Клиентская часть приложения реализована на фреймворке Vue 3, который обеспечивает реактивное управление состоянием и декларативное описание интерфейса. Vue 3 взаимодействует с серверной частью через HTTP и WebSocket, что позволяет обеспечить быструю реакцию интерфейса на изменения данных и поддержку реального времени.

- Web Server (HTTP/WebSocket):

В качестве основы для обработки клиент-серверного взаимодействия используются два веб-сервера. Первый обслуживает запросы, поступающие от клиентской части Vue 3, второй — запросы от poST Python интерпретатора. Это разделение улучшает производительность системы за счет распределения нагрузки и оптимизации потоков данных.

- Gunicorn + UvicornWorker (ASGI) [7]:

Gunicorn служит в качестве WSGI сервера, который использует UvicornWorker для поддержки асинхронной работы. Это позволяет использовать асинхронные и синхронные вызовы в рамках одного приложения. Uvicorn, ASGI

сервер, предоставляет высокопроизводительную асинхронную обработку запросов, что критически важно для поддержания низких задержек и высокой пропускной способности в интерактивных приложениях.

- FastAPI [8]:

FastAPI используется для создания веб-сервера, который обслуживает poST Python интерпретатор. Благодаря своей асинхронной природе и поддержке современных стандартов HTTP, FastAPI является идеальным выбором для создания RESTful API с поддержкой реального времени. Он обеспечивает эффективное взаимодействие между серверной логикой и poST интерпретатором, а также управляет динамическими запросами к состоянию виртуальных объектов управления.

2.2 State Management Pattern

Vuex [9] — это библиотека и шаблон управления состоянием для приложений Vue.js. Он предоставляет централизованное хранилище для всех компонентов в приложении и обеспечивает правила, гарантирующие, что состояние может изменяться только предсказуемым образом.

Шаблон управления состоянием (State Management Pattern) организует хранение и изменение состояния приложения таким образом, чтобы обеспечивать его централизованное и прозрачное управление.

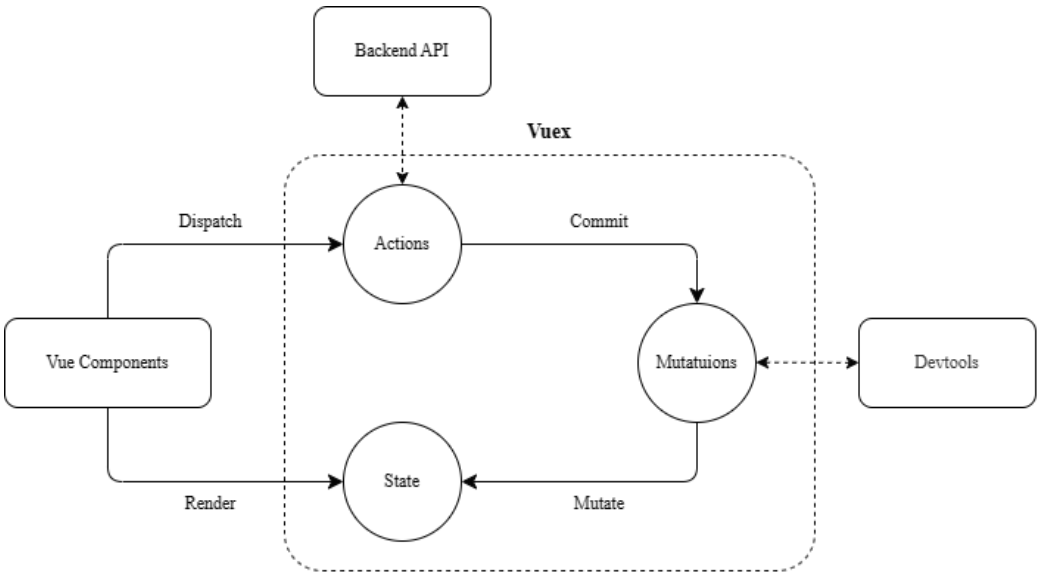

Рисунок 3. Vuex архитектура

Как показано на схеме, Vuex управляет состоянием приложения через несколько основных элементов:

- **State** (Состояние): Источник истины в приложении, который хранит все данные приложения.
- **View** (Представление): Компоненты Vue, которые отображают состояние в пользовательский интерфейс.
- **Actions** (Действия): Методы, которые вызываются из компонентов Vue для выполнения асинхронных операций или для действий, требующих взаимодействия с бэкендом.
- **Mutations** (Мутации): Функции, которые непосредственно изменяют состояние. Мутации вызываются действиями и должны быть синхронными.

Принцип работы

Компоненты Vue инициируют действия (actions), отправляя через метод dispatch запросы на изменение состояния.

Действия обрабатывают эти запросы, могут выполнять асинхронные операции и затем коммитят мутации через метод commit.

Мутации изменяют состояние в централизованном хранилище.

Изменения в состоянии автоматически отражаются в компонентах, которые зависят от этого состояния, благодаря реактивности Vue.

Этот процесс обеспечивает централизованное и однонаправленное управление потоком данных, что упрощает отладку и тестирование, а также повышает надежность приложения.

Обоснование использования

В нашем случае использование Vuex особенно удобно, так как оно позволяет эффективно инкапсулировать работу с данными.

Даже если в будущем механизмы получения или отправки данных изменятся, или потребуется изменение логики хранения данных, это не повлияет на компоненты приложения, которые используют эти данные. Благодаря Vuex, изменения могут быть локализованы в рамках действий (actions) и мутаций (mutations), без необходимости переписывать код компонентов или вносить изменения во многих местах программы. Это значительно упрощает поддержку кода и его масштабирование, делая приложение более устойчивым к изменениям и развитию функционала.

Таким образом, Vuex не только улучшает управление состоянием приложения, но и повышает его адаптивность и готовность к будущим модификациям, что является ключевым аспектом в долгосрочной перспективе разработки.

2.3 Composition API

Composition API представляет собой мощный инструмент в экосистеме Vue.js, который был введен в версии 3.0. Этот API предлагает альтернативный и более гибкий способ создания и организации компонентов по сравнению с традиционным Options API. В нашем проекте применение Composition API обосновывается несколькими ключевыми аспектами:

1. **Повышение читаемости и поддерживаемости кода:** Composition API улучшает структурирование кода, позволяя группировать связанный функционал по логическим блокам, а не разбрасывать его по различным опциям объекта компонента. Это делает компоненты более понятными и упрощает их тестирование и поддержку.

2. **Повторное использование логики:** API упрощает создание и переиспользование логики между компонентами без необходимости полагаться на механизмы, такие как миксины или высшие порядковые компоненты, которые могут создавать скрытые зависимости и усложнять отладку.

3. **Лучшее управление состоянием:** В контексте использования Vuex и обширной модели данных, Composition API облегчает управление локальным и глобальным состоянием, позволяя разработчикам легко интегрировать и синхронизировать состояние в рамках компонентов.

Применение в проекте

В рамках проекта, Composition API используется для создания функциональных блоков, управляющих различными аспектами визуализации и интерактивности интерфейса пользователя. Например, управление пользовательскими взаимодействиями, обработка данных от сервера и управление внутренним состоянием компонентов реализованы с использованием реактивных возможностей API.

Глава 3. Реализация и практическая апробация

3.1 Выбор технологий

На основе сформулированных требований к модулю визуализации для облачного практикума по языку poST, был проведен анализ доступных технологий для определения наиболее подходящих решений, способных удовлетворить заданные критерии эффективности, совместимости и функциональности.

Основные технологические решения:

1. Frontend разработка:

- Использование Composition API вместе с фреймворком Vue.js для построения интерактивного пользовательского интерфейса. Vue.js был выбран за его реактивные возможности и эффективное управление состоянием компонентов, что критически важно для реализации динамически обновляемых интерфейсов. Composition API обеспечивает более гибкое и модульное строение компонентов, что упрощает поддержку и масштабирование приложения.
- Применение Vuex для управления состоянием приложения позволяет централизованно управлять всеми данными внутри приложения, что необходимо для синхронизации состояний между различными компонентами и поддержки сложных взаимодействий внутри системы.

2. Инструменты для визуального дизайна и интерактивности:

- Google Blockly [10] используется для создания визуально-блочного интерфейса программирования, который позволяет пользователям без опыта кодирования строить логику управления виртуальными объектами через графические блоки. Этот инструмент подходит для реализации спецификации поведения объектов на основе событий и временных интервалов.
- Fabric.js выбран для работы с канвасом, позволяя разработчикам легко создавать и манипулировать графическими элементами на веб-странице. Fabric.js поддерживает как простые формы, так и сложные интерактивные объекты, что делает его идеальным для создания сложных визуализаций и пользовательских интерфейсов.

3. Backend и серверная инфраструктура:

- FastAPI выбран как основа для серверной части приложения из-за его высокой производительности, поддержки асинхронного программирования и встроенной поддержки стандартов OpenAPI. Это позволяет обеспечить быструю и надежную обработку клиентских запросов, а также упрощает интеграцию с другими системами и сервисами.
- ASGI/Uvicorn как асинхронный сервер, способный обрабатывать большое количество соединений параллельно, что жизненно важно для поддержки интерактивных веб-приложений с высокими требованиями к отзывчивости интерфейса.

4. Сборка и развертывание:

- Vite используется как современный инструмент сборки для веб-проектов, предоставляющий быструю пересборку модулей и горячую замену модулей (HMR). Это существенно ускоряет процесс разработки и тестирования, позволяя разработчикам видеть изменения в реальном времени без перезагрузки страницы.
- Docker [11] выбран для контейнеризации приложения и его зависимостей, что значительно упрощает процесс развертывания и управления инфраструктурой. Использование Docker позволяет создавать легко воспроизводимые среды разработки и продакшн-окружения, обеспечивая консистенцию между рабочими станциями разработчиков и серверами. Docker Compose дополнительно упрощает управление многоконтейнерными приложениями, позволяя описать и запустить множество контейнеров как единую службу с использованием одного файла конфигурации. Это делает процесс масштабирования и обновления компонентов приложения более управляемым и предсказуемым, обеспечивая эффективную координацию работы всех элементов системы.

3.2 Пользовательский интерфейс

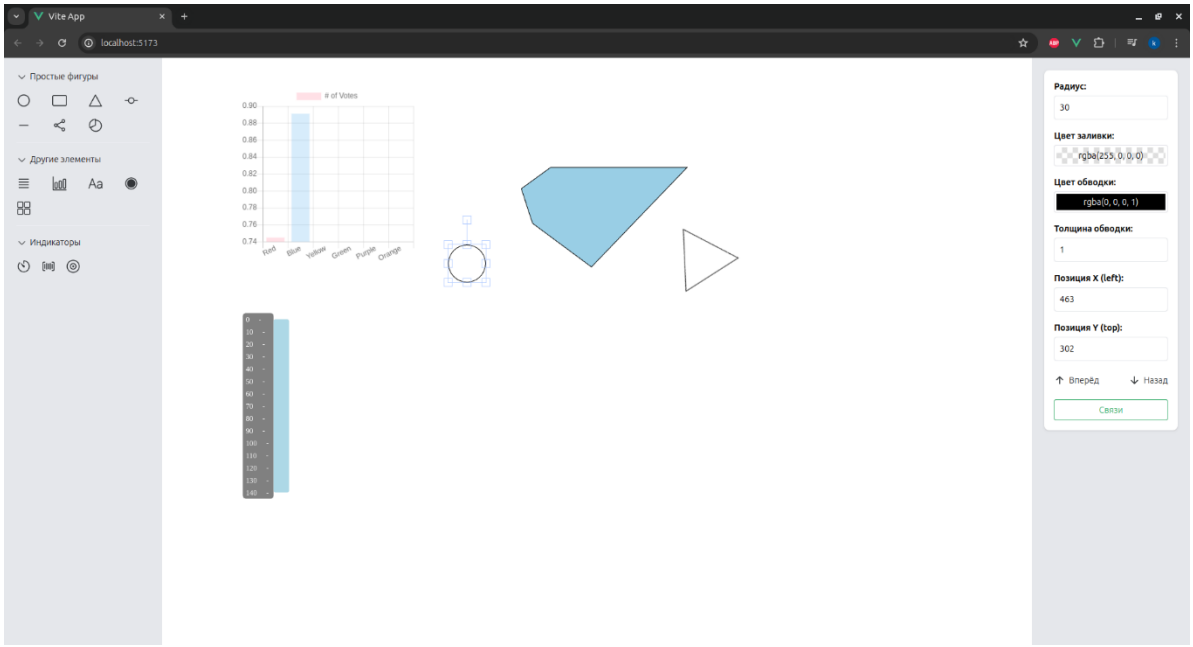

Рисунок 4. Пользовательский интерфейс

Интерфейс веб приложения состоит из трех основных частей – левой боковой панели, основного холста, и правой боковой панели.

Левая боковая панель отображает библиотеку элементов, представленных иконками.

Размещение элементов на холсте осуществляется через перетаскивание иконок в рабочее пространство.

Правая боковая панель отображает параметры и свойства текущего активного элемента. Отображение параметров обновляется реактивно и отображает текущие состояния в реальном времени. Пользователь может изменять все параметры элемента и визуальное отображение объекта будет так же реактивно изменятся.

Также правая боковая панель содержит кнопку «Связи», после нажатия на которую отобразится всплывающее окно (Рис. 5)

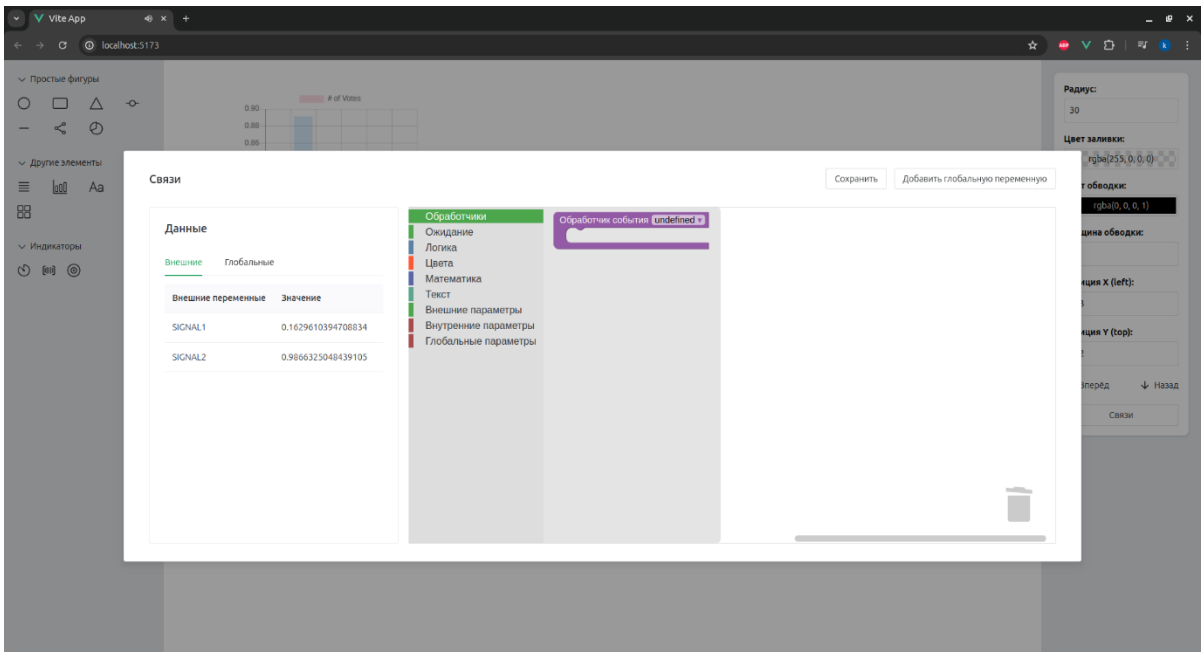

Рисунок 5. Всплывающее окно "Связи"

Окно содержит разбито на две части, левая часть представляет из себя меню с двумя вкладками, на каждой вкладке в виде таблицы отображаются текущие значения глобальных внутренних переменных и внешних в реальном времени.

Глобальные внутренние переменные – отвечают за глобальные состояния в локальном контексте веб приложения, и выступают в роли вспомогательного функционала. Доступ к ним имеют все элементы как на чтение, так и на редактирование. Предполагается, что данные параметры будут использоваться для передачи информации между элементами для имплементации нетривиальной логики визуализации, когда стандартных средств обработчиков не хватает.

Внешние переменные – отвечают за хранение информации, поступающей от веб сервера. Доступ к ним имеют все элементы, но только на чтение.

На правой стороне окна расположено рабочее пространство визуально-блочной событийно-ориентированной среды программирования, используемой

для задания логики поведения конкретного элемента в ответ на некоторые события.

Если блоки расположены без явного указания обработчика, то они будут вызываться при каждом обновлении внешних параметров, т.е. после каждого нового пакета данных, приходящих от сервера.

Блоки так же можно установить внутрь конкретного обработчика и указать определенное событие в ответ на которое будет выполнена логика из обработчика.

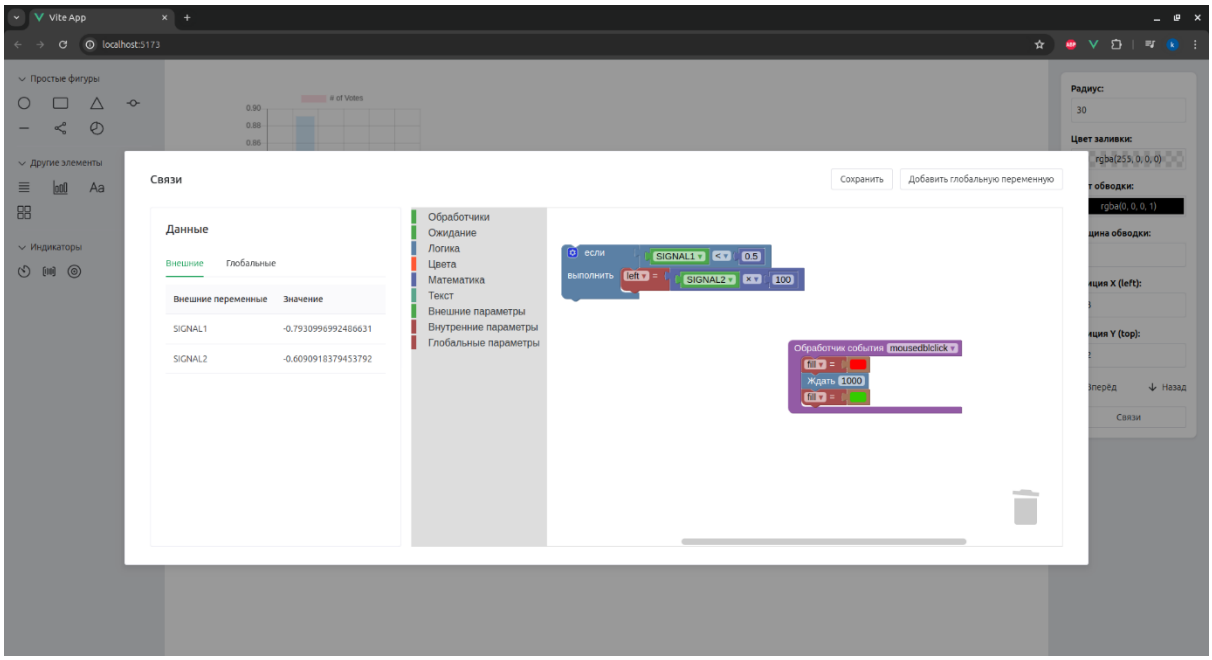

Рисунок 6. Рабочее пространство визуального программирования

На Рис. 6 представлен пример программы, задающей поведение элемента с двумя обработчиками – стандартным (срабатывающим в ответ на обновление внешних параметров), и пользовательским (срабатывающим в ответ на пользовательское событие, в данном случае на двойное нажатие мышью по объекту).

Стандартный обработчик изменяет положение элемента в пространстве по OY, если полученный от веб сервера параметр *SIGNAL1* содержит значение меньше, чем 0,5.

Пользовательский обработчик изменяет цвет элемента сначала на красный, а после ожидания в 1000 мс на зеленый, при двойном нажатии на элемент.

3.3 Структура и реализация графических компонент

Каждый графический элемент в системе представляет собой отдельный класс, расширяющий базовые возможности стандартных элементов Fabric.js. В Приложении Б представлен пример класса CircleShape, который расширяет стандартный класс fabric.Circle. В этом классе реализованы дополнительные методы и свойства, для сохранения кода на обработку событий, управления событиями и сериализации состояний.

Классы графических объектов интегрированы с хранилищем Vuex, что позволяет им реагировать на изменения глобальных переменных состояния приложения. Использование функции watch из Vue.js позволяет объектам автоматически обновлять свои свойства в ответ на изменения в хранилище, что необходимо для реализации реактивного поведения элементов управления.

Графические объекты обладают способностью обрабатывать различные события, такие как перемещение, изменение размера, наведение мыши и другие. Обработчики событий динамически назначаются и управляются через свойства класса, что обеспечивает гибкое управление поведением объектов в зависимости от контекста использования.

Для каждого графического элемента разработаны типовые Vue.js компоненты, позволяющие пользователям настраивать параметры объектов, такие как

размер, цвет, положение и другие. Эти компоненты взаимодействуют с экземплярами классов через двустороннюю привязку данных и обеспечивают обновление холста в ответ на пользовательские действия.

3.4 Структура и реализация рабочего пространства визуально-блочной среды программирования

Так как представляемого библиотекой Blockly базового функционала в полной мере недостаточно для полной реализации завяленных требований, были написаны расширения с новыми блоками и логикой их преобразования в исполняемый код.

Внешний вид всех блоков, их форма, цвет входные и выходные параметры, типы объектов с которыми они могут работать и прочее задаются в виде JSON конфигураций.

```
{
  "type": "handler",
  "message0": "Обработчик события %1 %2 %3 %4",
  "args0": [
    {
      "type": "input_dummy"
    },
    {
      "type": "field_variable",
      "name": "NAME",
      "variable": "undefined",

      "variableTypes": ["event"],
      "defaultType": "event",
```

```
    },
    {
      "type": "input_end_row"
    },
    {
      "type": "input_statement",
      "name": "NAME"
    }
  ],
  "inputsInline": false,
  "colour": 285,
  "tooltip": "",
  "helpUrl": ""
}
```

Листинг 1. Пример конфигурации отображения блока в библиотеке Blockly

Логика для преобразования конструкций из визуальный блоков задаётся в виде функций как показано в примере (см Листинг 3)

```
javascript.javascriptGenerator.forBlock['handler'] = (block, generator) => {
  const event_name = generator.nameDB_.getName(block.getFieldValue('NAME'), Blockly.VARIABLE_CATEGORY_NAME);
  const statements = generator.statementToCode(block, 'NAME');
  return `addEventListener[${event_name}]: ${statements.replace(/(\r\n|\n|\r)/gm, "")}`;
};
```

Листинг 2. Пример установки логики преобразования блока в исполняемый код

Во избежание проблем связанных с гонкой данных, например в случае, когда один обработчик изменил состояние какой-то переменной, а другой до сих пор работает со старым значением, весь сгенерированный код разбивается на

атомарные операции и после выполнения каждой возвращает промежуточные значения переменных через оператор *yield*, с которыми он работает.

```
this.returnStatement = `yield { ${this.shape.getProperties().map(varName => `${varName}: ${varName}`).join(", ")} };`;
```

```
javascript.javascriptGenerator.forBlock['internal_variable_set'] = (block, generator) => {
  const variable_name = generator.nameDB_.getName(block.getFieldValue('VAR'), Blockly.VARIABLE_CATEGORY_NAME);
  const value = generator.valueToCode(block, 'VALUE', javascript.Order.ASSIGNMENT) || '0';
  return `${variable_name} = ${value};\n${this.returnStatement}\n`;
};
```

Листинг 3. Пример установки логики преобразования блока переменных в исполняемый код

В результате всех преобразований сгенерированный код представляет из себя асинхронные функции генераторы, которые обрабатываются в классах элементов подобным образом

```
watch(() => globalVariableStore.getters['GET_GLOBAL_VARIABLES'], (newVal, oldVal) => {
  (async () => {
    if (this.dynamicFunction) {
      for (const key in await this.dynamicFunction.call(this, combinedParams)) {
      }
    }
  })();
}, {deep: true});
```

Листинг 4. Пример обработки результатов асинхронного генератора

3.5 Сериализация, хранение, импорт и экспорт данных

Весь сгенерированный код хранится в полях объектов элементов визуализации в виде строк и скомпилированных функций, что позволяет быстро запускать его при необходимости, а также иметь представление о его содержимом при экспорте данных.

Структура рабочего пространства Blockly так же хранится внутри объектов элементов визуализации и имеет встроенную функциональность социализации, предоставляемую библиотекой, что позволяет восстанавливать рабочее пространство при сворачивании всплывающего окна и очистки контекста.

При социализации, общего рабочего пространства, содержащего все элементы, получается список из JSON элементов, следующего вида:


```
{
 "objects": [
  {
   "type": "circle",
   "originX": "left",
   "originY": "top",
   "left": -0.76,
   "top": 205,
   "width": 60,
   "height": 60,
   "fill": "rgba(255, 0, 0, 0)",
   "dynamicFunctionSerialized": "cGltcGdhbmctNGV2ZXI=",
   "eventHandlerSerialized": "YmVsaWdrdG9oYQ==",
   "blocklyWorkspace": {
    "blocks": [
     {
      "languageVersion": 0,
```

```
        "blocks": [
          {
            "type": "internal_variable_set",
            "id": "KpWC3+GeD;4NC@N.Tnc9",
            "inputs": {}
          }
        ]
      }
    ],
    "variables": [
      {
        "name": "SIGNAL1",
        "id": "J6!NDx+=qV{QfWF!_b7,",
        "type": "external"
      }
    ]
  }
 }
}
```


Листинг 5. Пример сериализованного элемента

Каждый JSON объект содержит в себе информацию:

- Рабочего пространства Blockly, с конфигурацией всех блоков и переменных
- Внутренних параметров элемента, включая пользовательские и стандартные
- Сериализованые и закодированные в Base64 функции обработчиков

Такой подход позволяет сохранять сложные проекты в компактном виде и гарантирует быстрое восстановление рабочей среды при переносе на другие системы или платформы.

ЗАКЛЮЧЕНИЕ

Разработан комплекс визуализации для облачных практикумов по языку poST, интегрированный с системой виртуализации управляющих алгоритмов. Этот комплекс предоставляет возможность визуального проектирования и интерактивной визуализации моделей систем управления с динамической связью переменных, что улучшает восприятие и понимание процессов управления.

Созданный инструмент упрощает обучение разработке ПО систем управления. Графический редактор позволяет преподавателям и студентам визуализировать и тестировать управляющие алгоритмы, ускоряя процесс обучения.

Продолжается разработка и интеграция компонентов. Планируются тестирование и подготовка к внедрению в образовательный процесс.

Выпускная квалификационная работа выполнена мной самостоятельно и с соблюдением правил профессиональной этики. Все использованные в работе материалы и заимствованные принципиальные положения (концепции) из опубликованной научной литературы и других источников имеют ссылки на них. Я несу ответственность за приведенные данные и сделанные выводы.

Я ознакомлен с программой государственной итоговой аттестации, согласно которой обнаружение плагиата, фальсификации данных и ложного цитирования является основанием для не допуска к защите выпускной квалификационной работы и выставления оценки «неудовлетворительно».

Ли Константин Алексеевич _____

ФИО студента *Подпись студента*

« ____ »_____2024 г.

(заполняется от руки)

Список литературы

1. Зюбин В. Е. Итерационная разработка управляющих алгоритмов на основе имитационного моделирования объекта управления //Автоматизация в промышленности. – 2010. – №. 11. – С. 43-48.

2. Kodosky J. LabVIEW //Proceedings of the ACM on Programming Languages. – 2020. – Т. 4. – №. HOPL. – С. 1-54.

3. Hanssen D. H. Programmable logic controllers: a practical approach to IEC 61131-3 using CODESYS. – John Wiley & Sons, 2015.

4. Масеевский А. М., рук. Зюбин В. Е, Разработка веб-технологии виртуализации ПЛК средствами Python-интерпретатора для исполнения poST-программ // Выпускная квалификационная работа бакалавра, ФИТ НГУ, 2023, – 47 с.

5. Витченко В. А. Разработка WYSIWYG-редактора для модуля динамической верификации процесс-ориентированных алгоритмов управления //МНСК-2020. – 2020. – С. 168-168.

6. Pablo D. Vue.js 3 - Design Patterns and Best Practices – 2014.

7. Novikau A. Evaluative Comparison of ASGI Web Servers: A Systematic Review.

8. Lathkar M. High-Performance Web Apps with FastAPI. – Apress, 2023. – С. 65-92.

9. Wu B. Analyze and Compare the State Management Patterns of VUE //2023 IEEE International Conference on Image Processing and Computer Applications (ICIPCA). – IEEE, 2023. – С. 1279-1282.

10. Ashrov A. et al. A use-case for behavioral programming: An architecture in JavaScript and Blockly for interactive applications with cross-cutting scenarios //Science of Computer Programming. – 2015. – Т. 98. – С. 268-292.

11. Docker I. Docker //lınea].[Junio de 2017]. Disponible en: https://www. docker. com/what-docker. – 2020.

ПРИЛОЖЕНИЕ А

Скриншоты страниц разработанного веб-интерфейса приложения

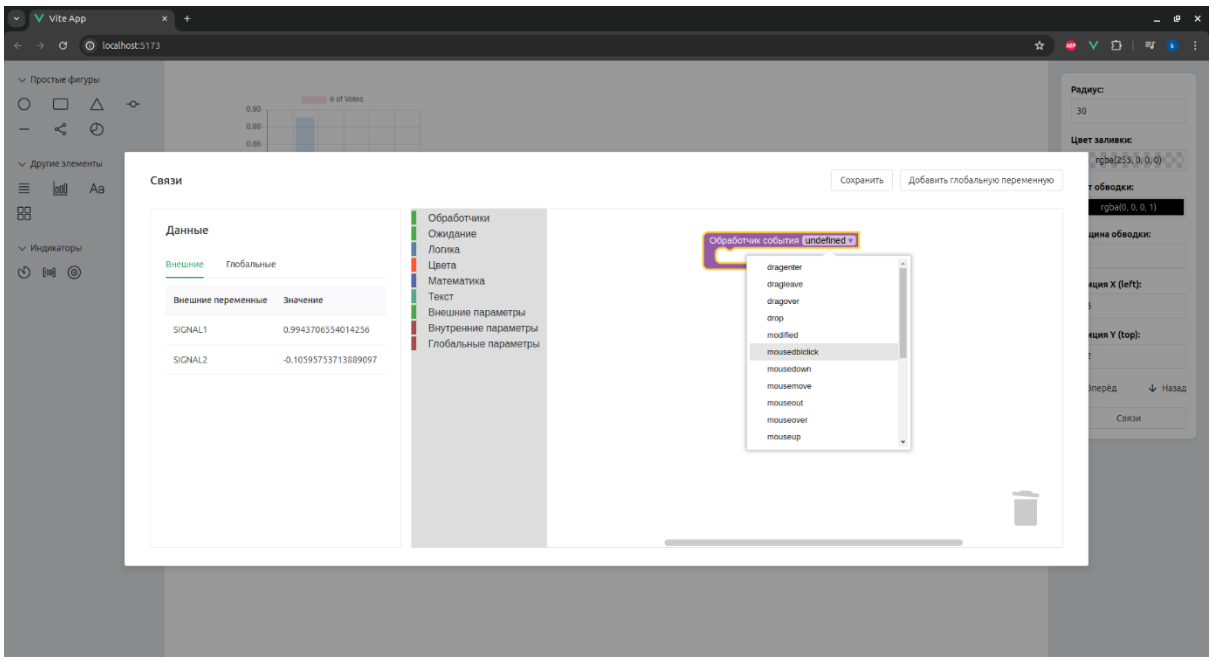

Рисунок 7. Пример возможных типов обработчика

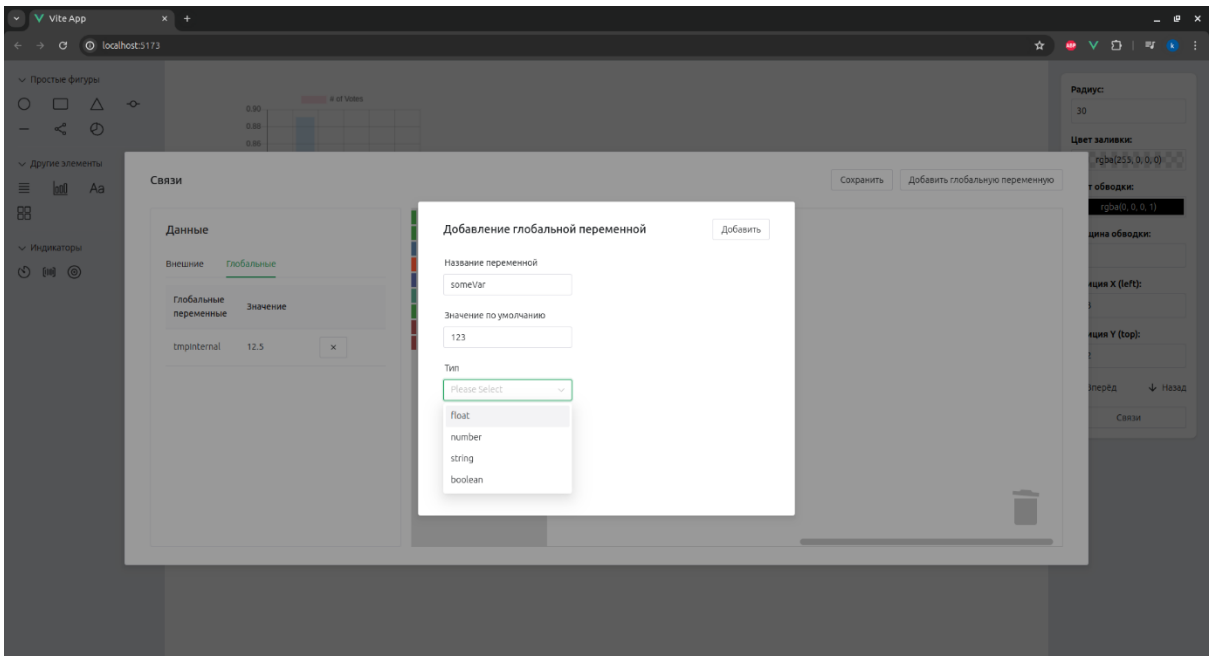

Рисунок 8. Окно добавления новой глобальной переменной

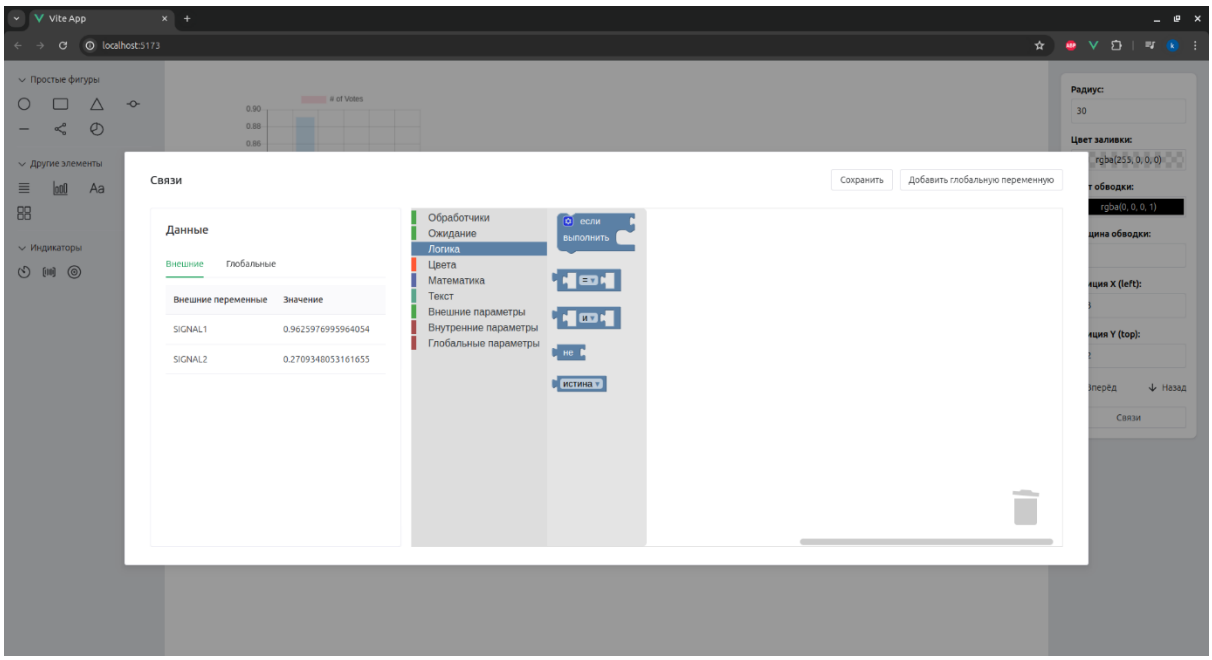

Рисунок 9. Логические блоки

ПРИЛОЖЕНИЕ Б

Пример расширения стандартного класса fabric.js

```
import {fabric} from 'fabric';
import {globalVariableStore} from "@/store/index.js";
import {watch} from "vue";

export class CircleShape extends fabric.Circle {
    private canvas: any;
    private dynamicFunction: Function | null = null;
    private blocklyWorkspace: any = null;

    private eventHandlers: Record<string, any> = {
        moving: null,
        scaling: null,
        rotating: null,
        skewing: null,
        resizing: null,
        mouseup: null,
        mousedown: null,
        mousemove: null,
        mousedblclick: null,
        mousewheel: null,
        mouseover: null,
        mouseout: null,
        drop: null,
        dragover: null,
        dragenter: null,
        dragleave: null,
        modified: null,
        over: null,
```

```
};
private properties: string[] = [
    "left",
    "top",
    "fill"
];

constructor(canvas: any, x: number, y: number) {
    super({
        radius: 30,
        fill: 'rgba(255, 0, 0, 0)',
        stroke: 'rgba(0, 0, 0, 1)',
        strokeWidth: 1,
        strokeUniform: true,
        left: x,
        top: y,
    });
    this.canvas = canvas;
    this.canvas.add(this);
    this.canvas.requestRenderAll();

    Object.keys(this.eventHandlers).forEach(key => {
        this.on(key, (options: any) => {
            (async () => {
                await this.handleEvent(key, options);
            })();
        });
    });
```

```
            watch(() => globalVariableStore.getters['GET_GLOBAL_VARIABLES'],
(newVal, oldVal) => {
              ...
            }, {deep: true});
        }

        public setEventHandlers(eventHandlers: Record<string, any>) {
            ...
        }


        async handleEvent(eventName: string, options: any) {
            if (this.eventHandlers.hasOwnProperty(eventName) &&
this.eventHandlers[eventName] !== null) {
                const global_variables = globalVariableStore.get-
ters['GET_GLOBAL_VARIABLES'];
                let params = this.simplifyParams(global_variables);
                let props = this.getPropertiesAsDictionary()
                let combinedParams = {...params, ...props};

                const generator = this.eventHandlers[eventName].call(this, combined-
Params);

                for await (const result of generator) {

                  for (const key in result) {
                    const value = result[key];
                    if (value) {
                      this.set(key.toLowerCase(), value)
                    }
```

```
                }
                this.canvas.requestRenderAll();
            }
        }
    }

    toObject(propertiesToInclude: string[] = []) {
        ...
    }
}
```

ПРИЛОЖЕНИЕ В

МОДУЛЬ ВИЗУАЛИЗАЦИИ ДЛЯ ОБЛАЧНОГО ПРАКТИКУМА ПО ЯЗЫКУ POST

РУКОВОДСТВО ОПЕРАТОРА

Листов 9

Новосибирск, 2024

Содержание

Аннотация

В данном программном документе приведено руководство оператора по применению и эксплуатации модуля динамического отображения, редактирования и экспорта для облачного практикума по языку poST.

В разделе «Назначение программы» указаны сведения о функциональном и эксплуатационном назначении программы, а также перечислены основные функции, которые обеспечивает программное обеспечение.

Раздел «Условия выполнения программы» содержит требования к аппаратному и программному обеспечению, а также навыки, необходимые для оператора программы.

Раздел «Выполнение программы» описывает последовательность действий оператора, которые необходимы для загрузки, запуска, выполнения и завершения работы программы, а также основные операции, такие как работа с графическим редактором, динамическое отображение данных и экспорт информации.

1. Назначение программы

1.1 Функциональное назначение

Программа предназначена для создания, редактирования и визуализации графических элементов, представляющих физические и управляющие компоненты системы автоматизации. Основные функции включают в себя построение графических моделей, настройку логики управления объектами через визуально-блочные схемы, а также мониторинг и анализ данных в реальном времени.

1.2 Эксплуатационное назначение

Программа используется в образовательных целях для обучения студентов программированию и автоматизации с помощью языка poST. Она позволяет преподавателям и студентам создавать, редактировать, визуализировать и тестировать управляющие алгоритмы, создавая интерактивные учебные материалы и задания. Программа интегрируется с облачной платформой, обеспечивая доступ к учебным материалам и инструментам визуализации в режиме онлайн.

1.3 Состав функций

1. Отображение и редактирование графических элементов:

- Создание и настройка графических элементов, представляющих физические и управляющие компоненты системы.
- Поддержка основных операций с графическими элементами, таких как перемещение, изменение размера, и изменение цвета.

3. Визуально-блочное программирование:

- Создание и настройка логики поведения элементов через визуально-блочные схемы на основе Google Blockly.
- Поддержка обработки событий и выполнение логики на основе событий и временных интервалов.

4. Динамическое отображение данных:

- Мониторинг и отображение данных в реальном времени.
- Обновление графических элементов в зависимости от поступающих данных.

5. Экспорт и импорт данных:

- Экспорт конфигураций и настроек элементов в формате JSON.
- Импорт данных и восстановление рабочей среды из экспортированных файлов.

2. Условия выполнения программы

2.1 Минимальный состав аппаратных средств

Для использования программного обеспечения необходимо наличие персонального компьютера, включающего в себя:

- Процессор на архитектуре x86 с частотой 2 ГГц или выше.
- Оперативную память объемом 4 ГБ или выше.
- Графический адаптер, поддерживающий отображение современных графических интерфейсов.
- Монитор с разрешением не менее 1920x1080 пикселей.
- Клавиатуру и мышь.
- Доступ в интернет со скоростью не менее 5 Мбит/с.

2.2 Минимальный состав программных средств

Для использования программного обеспечения необходима предустановка следующих программных средств:

- Операционная система:
- Windows 10 или выше.
- MacOS 10.14 или выше.
- Любая современная версия дистрибутива Linux с поддержкой графического интерфейса.
- Web-браузер, поддерживающий отображение динамических HTML5-страниц:
- Google Chrome версия 89 или выше.
- Mozilla Firefox версия 86 или выше.
- Microsoft Edge версия 89 или выше.
- Среда исполнения Docker для контейнеризации:
- Docker Engine версия 20.10 или выше.
- Docker Compose версия 1.29 или выше.

2.3 Требования к оператору

Для конечного оператора (пользователя) программного обеспечения предъявляются следующие требования:

- Базовые навыки работы с графическим пользовательским интерфейсом операционной системы.
- Умение использовать web-браузер для навигации и взаимодействия с веб-приложениями.
- Основные знания в области программирования и автоматизации.
- Наличие зарегистрированной учетной записи в системе, обеспечивающей доступ к облачному практикуму по языку poST.

3. Выполнение программы

3.1 Загрузка и запуск программы

1. Откройте web-браузер, поддерживающий HTML5, и перейдите на страницу облачного практикума по языку poST.
2. Введите данные учетной записи для авторизации в системе.
3. После успешной авторизации откройте модуль динамического отображения и редактирования, выбрав соответствующую опцию в меню.

3.2 Работа с графическим редактором

Работа с графическим редактором

1. Левая боковая панель:

- Отображает библиотеку элементов, представленных иконками.
- Для размещения элементов на холсте перетащите нужную иконку в рабочее пространство.

2. Основной холст

- Здесь размещаются и редактируются графические элементы.
- Используйте функции перемещения и изменения размеров элементов.

3. Правая боковая панель

- Параметры обновляются реактивно и отображают текущие состояния в реальном времени.
- Пользователь может изменять параметры элемента, и визуальное отображение объекта будет изменяться автоматически.
- Отображает параметры и свойства текущего активного элемента

4. Кнопка "Связи"

- Нажмите на кнопку "Связи" для отображения всплывающего окна.
- Всплывающее окно содержит две части: меню с вкладками и рабочее пространство визуально-блочной программирования.

Динамическое отображение данных

1. Глобальные внутренние переменные

- Отвечают за глобальные состояния в локальном контексте веб-приложения.
- Используются для передачи информации между элементами для реализации сложной логики визуализации.
- Доступны всем элементам как для чтения, так и для редактирования.

2. Внешние переменные

- Отвечают за хранение информации, поступающей от веб-сервера.
- Доступны всем элементам только для чтения.

3. Рабочее пространство визуально-блочной событийно-ориентированной среды программирования

- Используется для задания логики поведения элементов в ответ на события.
- Блоки могут быть установлены внутрь обработчика для выполнения логики в ответ на определенные события.
- Если блоки расположены без явного указания обработчика, они будут вызываться при каждом обновлении внешних параметров.

3.3 Завершение программы

Для закрытия программы оператору необходимо закрыть вкладку браузера, в котором открыто приложение.

4. Лист регистрации изменений

Лист регистрации изменений									
Номера листов (страниц)					Всего листов (страниц) в документе	№ документа	Входящий № сопроводительного документа	Подпись	Дата
Номер изм.	измененных	замененных	новых	аннулированных					

Таблица 1. Лист регистрации изменений в программном документе «Руководство оператора»

ПРИЛОЖЕНИЕ Г

МОДУЛЬ ВИЗУАЛИЗАЦИИ ДЛЯ ОБЛАЧНОГО ПРАКТИКУМА ПО ЯЗЫКУ POST

ОПИСАНИЕ ПРОГРАММЫ

Содержание

Аннотация

В данном программном документе приведено описание модуля визуализации для облачного практикума по языку poST. Программный модуль предназначен для создания, редактирования и визуализации графических элементов, представляющих физические и управляющие компоненты систем, а также их динамическое взаимодействие.

В разделе «Общие сведения» указаны сведения о программном обеспечении, необходимом для функционирования приложения и языке программирования.

В разделе «Назначение программы» указаны сведения о назначении программы и ее функциональности.

В разделе «Описание логической структуры» указаны сведения об алгоритме программы, используемых методах и структуре программы.

В разделах «Входные данные» и «Выходные данные» указаны форматы и особенности организации входных и выходных данных.

1. Общие сведения

1.1 Обозначение и наименование программы

Наименование программы: Модуль визуализации для облачного практикума по языку poST.

1.2 Программное обеспечение, необходимое для функционирования программы

Для работы модуля необходимы следующие компоненты:

- Node.js версии 14.0 или выше.
- Vue.js для разработки интерфейса.
- FastAPI для серверной части.
- Vite для сборки проекта.

1.3 Языки программирования

Исходным языком программного обеспечения редактора является TypeScript. Основным средством разработки является интегрированная среда WebStorm от компании Jetbrains. Для отладки использовались средства браузера — Google Chrome DevTools.

Серверная часть приложения разработана на языке Python с использованием фреймворка FastAPI. Интегрированная среда разработки для серверной части — PyCharm от компании Jetbrains.

2. Назначение программы

2.1 Функциональное назначение программы

Программное обеспечение предназначено для создания, редактирования и визуализации графических элементов, представляющих физические и управляющие компоненты систем. Основная цель - обеспечить интерактивное визуальное проектирование и динамическую визуализацию систем управления на языке poST в реальном времени.

2.2 Эксплуатационное назначение программы

Приложение предназначено для использования в образовательных целях, позволяя студентам и преподавателям обучаться разработке и тестированию управляющих алгоритмов. Оно помогает визуализировать процессы управления и взаимодействие компонентов системы, что способствует лучшему пониманию и усвоению учебного материала.

2.3 Состав функций

1. Создание графических элементов: Пользователь может создавать различные графические элементы, такие как блоки, линии и фигуры, представляющие компоненты системы.
2. Редактирование элементов: Возможность изменения параметров и свойств графических элементов, включая размер, цвет, положение и другие характеристики.
3. Визуализация: Динамическое отображение изменений в реальном времени в ответ на изменения параметров и событий.
4. Интерактивное взаимодействие: Поддержка drag-and-drop для размещения элементов на рабочем пространстве и настройки их поведения.
5. Интеграция с серверной частью: Получение и отправка данных на сервер для синхронизации состояния и обработки команд.
6. Экспорт данных: Сохранение конфигурации и параметров проекта в формате, удобном для дальнейшего использования и анализа.

2.4 Минимальный состав аппаратных средств

Минимальный состав аппаратных средств Для использования программного обеспечения необходимо наличие персонального компьютера, включающего в себя:

1. Процессор на архитектуре x86 с частотой 2 ГГц или выше;
2. Оперативную память объемом 4 Гб или выше;
3. Графический адаптер;
4. Монитор, клавиатура, мышь;
5. Доступ в интернет.

3. Описание логической структуры

3.1 Алгоритм программы

В модуле визуализации для облачного практикума по языку poST реализована возможность создания, редактирования и визуализации графических элементов, представляющих компоненты систем управления. Алгоритм работы программы включает следующие этапы:

1. Запуск программы и инициализация: При запуске программы и при каждом изменении состояния приложение получает данные о текущем состоянии системы от сервера.
2. Обновление состояния интерфейса: Пользователь взаимодействует с интерфейсом через перетаскивание элементов и изменение их параметров. Изменения фиксируются и передаются в центральное хранилище состояния (Vuex).
3. Обработка действий пользователя: Компоненты, получившие внешний сигнал, генерируют Action, содержащий информацию о произошедшем событии (например, изменение параметра или добавление нового элемента). Эти действия передаются в центральное хранилище состояния.
4. Обновление состояния приложения: Специальные функции обрабатывают Action и обновляют текущее состояние приложения. Новое состояние синхронизируется с сервером при необходимости.
5. Рендеринг интерфейса: Компоненты интерфейса обновляются в соответствии с изменениями состояния, обеспечивая реактивное отображение и взаимодействие.

3.2 Структура программы

Код программы можно разделить на два компонента: backend часть и frontend часть.

Backend часть реализована на основе FastAPI и отвечает за обработку запросов от клиента, управление данными и взаимодействие с базой данных. Основные функции backend части включают:

Обработка запросов: Обработка клиентских запросов на получение и отправку данных.

Управление состоянием системы: Синхронизация данных между клиентом и сервером, обработка событий и выполнение серверной логики.

Интеграция с другими системами: Поддержка стандартов OpenAPI для упрощения интеграции с другими системами и сервисами.

Frontend часть разработана с использованием Vue.js и включает в себя логику взаимодействия пользователя с приложением и формирование визуальных данных. Основные компоненты frontend части включают:

Интерфейс пользователя: Интерфейс состоит из трех основных частей – левой боковой панели, основного холста и правой боковой панели.

Левая боковая панель: Отображает библиотеку элементов, представленных иконками. Размещение элементов на холсте осуществляется через перетаскивание иконок в рабочее пространство.

Основной холст: Рабочее пространство для размещения и взаимодействия с графическими элементами.

Правая боковая панель: Отображает параметры и свойства текущего активного элемента, которые можно изменять в реальном времени.

Визуально-блочная среда программирования: Google Blockly используется для создания визуально-блочного интерфейса программирования, позволяющего пользователям строить логику управления виртуальными объектами через графические блоки.

Графические элементы: Реализованы с использованием Fabric.js для работы с канвасом. Каждый графический элемент представляет собой отдельный класс, расширяющий базовые возможности стандартных элементов Fabric.js.

Управление состоянием: Синхронизация состояния между компонентами и сервером через Vuex.

Взаимодействие между backend и frontend частями происходит посредством протокола HTTP, что обеспечивает надежную и эффективную передачу данных.

4. Входные данные

Входными данными для модуля визуализации являются:

Пользовательские данные:

- Действия пользователя: Ввод данных с клавиатуры, клики и перетаскивание элементов с помощью мыши.
- Параметры и свойства элементов: Ввод и изменение параметров графических элементов, таких как размер, цвет, положение и другие характеристики.
- Загрузка файлов: Изображения и другие медиафайлы для использования в визуализации.

Серверные данные:

- Параметры системы: Данные о текущем состоянии системы, получаемые от сервера.
- События и команды: События и команды, поступающие от серверной части для синхронизации состояния и управления графическими элементами.
- Данные визуального блочного программирования:

Конфигурация блоков: JSON-конфигурации блоков для Google Blockly, включая форму, цвет, входные и выходные параметры.

Логика блоков: Функции преобразования визуальных блоков в исполняемый код на JavaScript.

5. Выходные данные

Выходными данными для модуля визуализации являются:

Данные для пользователя:

- Обновления интерфейса: Реактивное отображение изменений в графических элементах и их параметрах в реальном времени.
- Экспортированные файлы: Сохраненные конфигурации и проекты в виде JSON и XML файлов для дальнейшего использования и анализа.
- Визуализация блоков: Сгенерированный код на основе визуальных блоков, выполняемый в реальном времени для управления графическими элементами.

Данные для сервера:

- Состояние системы: Обновленные данные о состоянии системы и параметрах графических элементов для синхронизации с серверной частью.
- События и команды: События и команды, генерируемые на клиентской части и отправляемые на сервер для обработки.
- Сериализация и хранение данных:

Сериализованные объекты: JSON-объекты, содержащие информацию о рабочих пространствах Blockly, конфигурации блоков и параметрах элементов.

Код обработчиков: Сохраненные функции обработчиков событий, закодированные в Base64, для последующего восстановления и выполнения.

6. Лист регистрации изменений

Лист регистрации изменений									
Номера листов (страниц)					Всего листов (страниц) в документе	№ документа	Входящий № сопроводительного документа	Подпись	Дата
Номер изм.	измененных	замененных	новых	аннулированных					

Таблица 2. Лист регистрации изменений в программном документе «Описание программы»