МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

РАБОТА ПРОВЕРЕНА

Рецензент
Специалист по защите информации
Научно – образовательного центра
«Информационная безопасность»
ФГАОУ ВО «ЮУрГУ (НИУ)»

_____ Баринов А.Е.

«___»_____ 2024 г.

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Разработка фронтенда интеллектуальной АСУ ТП обогащения руды ГОКа

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 02.04.02.2024.308-1388.ВКР

Научный руководитель,
доцент кафедры СП, к.ф.-м.н.
_____ К.И. Костромитин

Автор работы,
студент группы КЭ-220
_____ М.Е. Гаврилов

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы магистранта
студенту группы КЭ-220
Гаврилову Максиму Евгеньевичу,
обучающемуся по направлению
02.04.02 «Фундаментальная информатика и информационные технологии»
(магистерская программа «Машинное обучение и анализ больших данных»)

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 308/08)
Разработка фронтенда интеллектуальной АСУ ТП обогащения руды ГОКа.
2. **Срок сдачи студентом законченной работы:** 20.05.2024 г.
3. **Исходные данные к работе**

3.1. Типизированный язык программирования (надстройка над JavaScript) | TypeScript. [Электронный ресурс] URL: https://www.typescriptlang.org/ (дата обращения: 17.02.2024 г.).

3.2. React: Документация JavaScript-библиотеки. [Электронный ресурс] URL: https://ru.legacy.reactjs.org/docs/getting-started.html. (дата обращения: 09.01.2024 г.).

3.3. Apache Echarts: Библиотека визуализации JavaScript. [Электронный ресурс] URL: https://echarts.apache.org/en/option.html#title (дата обращения: 11.01.2024 г.).

3.4. Справочник по API | effector. [Электронный ресурс] URL: https://effector.dev/ru/api/effector-react/ (дата обращения: 22.01.2024 г.).

4. Перечень подлежащих разработке вопросов

4.1. Проектирование архитектуры и макета системы мониторинга.

4.2. Добавление функций для запроса данных с эндпоинта.

4.3. Разработка функциональных компонентов для выбора дат и смен.

4.4. Разработка меню мониторинга.

4.5. Реализация и настройка графиков библиотеки Echarts.

4.6. Протестировать систему мониторинга.

5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
доцент кафедры СП, к.ф.-м.н. К.И. Костромитин

Задание принял к исполнению М.Е. Гаврилов

ОГЛАВЛЕНИЕ

ГЛОССАРИЙ

1. *ГОК (Горно-обогатительный комплекс)* – предприятие, занимающееся добычей и первичной переработкой полезных ископаемых для получения концентрата [1].

2. *АСУ ТП (Автоматизированная система управления технологическими процессами)* – система для контроля и управления производственными процессами в реальном времени [2].

3. *Чекбокс* – элемент управления в пользовательском интерфейсе, позволяющий пользователю выбирать один или несколько параметров из предложенного набора [3].

4. *Мониторинг* – процесс наблюдения и отслеживания состояния системы или процессов в реальном времени с целью их оптимизации и контроля [4].

5. *Обогащение руды* – комплекс мероприятий по разделению руды на полезную часть (концентрат) и отходы (хвосты) [4].

6. *JSON (JavaScript Object Notation)* – текстовый формат обмена данными, основанный на JavaScript, часто используется для передачи данных между сервером и клиентом [4].

7. *API (Application Programming Interface)* – набор правил и спецификаций, позволяющий программам взаимодействовать друг с другом [5].

8. *Frontend* – клиентская часть приложения, обеспечивающая взаимодействие с пользователем через графический интерфейс [6].

9. *Backend* – серверная часть приложения, обеспечивающая обработку данных, аутентификацию пользователей и выполнение бизнес-логики [4].

ВВЕДЕНИЕ

Актуальность

Разработка фронтенда для интеллектуальной системы автоматизированного управления технологическими процессами (САУТП) обогащения руды ГОКа актуальна из-за необходимости повышения эффективности и снижения затрат в горнодобывающей отрасли. Внедрение современных технологий в интерфейсы пользователей позволяет улучшить контроль над производственными процессами, обеспечивая более высокую безопасность и экологическую устойчивость предприятий.

Современная горнодобывающая промышленность сталкивается с множеством вызовов, среди которых повышение эффективности производственных процессов, минимизация воздействия на окружающую среду и обеспечение безопасности труда. В условиях возрастающей конкуренции и строгих экологических норм, актуальность применения инновационных технологий в добыче и обработке руды возрастает. Разработка и внедрение интеллектуальных систем управления на горно-обогатительных комплексах отвечает этим требованиям, позволяя оптимизировать процессы, снизить затраты и повысить общую производительность.

ГОКи испытывают постоянное давление в плане снижения производственных затрат и увеличения объемов производства. Традиционные методы управления уже не могут полностью справиться с этой задачей, так как они не обладают достаточной гибкостью и адаптивностью к изменяющимся условиям эксплуатации. Интеллектуальные системы управления, использующие данные в реальном времени и алгоритмы машинного обучения, могут значительно улучшить ситуацию. Они способны анализировать большие объемы данных, быстро реагируя на изменения в процессе обогащения и предотвращая ненужные простои и потери ресурсов.

Вторым значимым аспектом является стремление к минимизации воздействия на окружающую среду. ГОКи часто становятся объектами критики со стороны экологических организаций из-за высокого уровня за-

грязнения, которое они производят. Использование интеллектуальных систем позволяет не только оптимизировать потребление ресурсов, таких как вода и энергия, но и уменьшить количество отходов и выбросов за счет более точного контроля за процессами.

Третья актуальная проблема – это обеспечение безопасности на производстве. Горнодобывающая отрасль традиционно ассоциируется с высокими рисками для здоровья и жизни работников. Применение современных технологий и автоматизация контроля за производственными процессами могут существенно снизить эти риски. Интеллектуальные системы способны предсказывать потенциальные аварийные ситуации, анализируя данные с датчиков в реальном времени, что позволяет предпринимать меры по их предотвращению еще до того, как произойдет авария.

Данная система будет внедряться в Свердловский горно-обогатительный комбинат (КГОК). КГОК входит в пятерку крупнейших в России горнорудных предприятий. Разрабатывает Гусевогорское и Собственно-Качканарское месторождения титаномагнетитовых железных руд с содержанием ванадия. На нем добывается более 60 млн тонн руды из пяти карьеров.

Конечный продукт предприятия – агломерат и окатыши, которые являются железорудным сырьем для производства ванадиевого чугуна в доменных печах на ЕВРАЗ Нижнетагильском металлургическом комбинате. С начала работы Свердловского ГОКа в карьерах пробурили более 41 млн погонных метров скважин [11].

Разработка и внедрение интеллектуальных систем управления в горно-обогатительных комплексах – это не только технологическая необходимость, но и экономическая выгода. Такие системы позволяют достигать высоких результатов в оптимизации производственных процессов, уменьшении экологического воздействия и повышении уровня безопасности на предприятиях. Это делает их разработку и внедрение крайне актуальным направлением в современных условиях эксплуатации ГОКов.

Постановка задачи

Целью выпускной квалификационной работы является разработка фронтенда интеллектуальной САУТП обогащение руды ГОКа. Для достижения поставленной цели необходимо решить следующие задачи:

1) проектирование архитектуры и макета системы мониторинга;
2) добавление функций для запроса данных с эндпоинтов;
3) разработка функциональных компонентов для выбора дат и смен;
4) разработка меню мониторинга;
5) реализация и настройка графиков библиотеки Echarts;
6) Тестирование системы мониторинга.

Структура и содержание работы

Работа состоит из введения, пяти глав, заключения и списка литературы. Объем работы составляет 46 страниц, объем списка литературы – 20 источников.

В первой главе описывается анализ предметной области, обогащение и управление процессами, а также обзор аналогов.

Вторая глава посвящена функциональным и нефункциональным требованиям и диаграмме вариантов использования.

В третьей главе представлен обзор технологического стека, применяемого в проекте, а также обоснование выбора конкретных технологий.

Четвертая глава посвящена разработке веб-интерфейса, описанию как устроен фронтэнд, взаимодействие компонентов между собой.

Пятая глава посвящена тестированию системы, в ней рассматриваются методы и результаты тестирования адаптивности, кросс-браузерной совместимости и функциональности приложения.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Обзор горно-обогатительного комплекса (ГОК) и процессов обогащения руды

Горно-обогатительный комплекс (ГОК) – это предприятие, занимающееся добычей и первичной переработкой полезных ископаемых с целью получения концентрата, пригодного для дальнейшей промышленной обработки. Процесс обогащения руды является ключевым на ГОКе, поскольку он позволяет значительно повысить содержание полезного компонента в руде, уменьшая долю пустой породы.

Процессы обогащения

Обогащение руды – это комплекс мероприятий по разделению руды на полезную часть (концентрат) и отходы (хвосты). Основные методы обогащения включают дробление, измельчение, флотацию, гравитационное обогащение, магнитную и электростатическую сепарацию. Выбор метода обогащения зависит от типа руды, физических и химических свойств полезного компонента и пустой породы.

Управление процессами

Эффективное управление процессами обогащения руды критически важно для ГОКа, поскольку от этого напрямую зависят как экономическая эффективность предприятия, так и экологическая безопасность региона. Внедрение современных информационных технологий и систем управления позволяет оптимизировать процессы, повысить точность контроля за ходом обогащения и уменьшить воздействие на окружающую среду.

Информационные технологии на ГОКе

Использование информационных технологий, включая системы мониторинга и автоматизации, играет важную роль в оптимизации процессов на ГОКе. Современные технологии позволяют собирать и анализировать данные в реальном времени, прогнозировать эффективность процессов обогащения, а также предотвращать аварийные ситуации.

Проблемы и вызовы

Основные проблемы, с которыми сталкиваются ГОКи, включают износ оборудования, высокие энергозатраты, необходимость снижения воздействия на окружающую среду и необходимость повышения общей эффективности производства. Решение этих проблем требует комплексного подхода, включая модернизацию оборудования, внедрение новых технологий и улучшение процессов управления.

Анализ горно-обогатительного комплекса и процессов обогащения руды показывает, что использование современных информационных технологий и систем управления может значительно повысить эффективность работы ГОКа, снизить затраты и минимизировать негативное воздействие на окружающую среду. Важно не только внедрять новые технологии, но и постоянно совершенствовать их.

1.2. Актуальность разработки системы для ГОКа

В современных условиях эксплуатации горно-обогатительных комплексов (ГОКов) появляется растущая потребность в повышении эффективности процессов обогащения руды. Это обусловлено не только экономическими аспектами, такими как увеличение прибыли и сокращение затрат на производство, но и экологическими требованиями, направленными на минимизацию вредного воздействия на окружающую среду. В этом контексте разработка и внедрение интеллектуальных систем управления производственными процессами становится не просто актуальной задачей, но и необходимостью для современных ГОКов [18].

Интеллектуальные системы управления могут кардинально изменить подходы к организации процессов на ГОКах. Использование таких систем позволяет не только автоматизировать рутинные операции, но и обеспечивает возможность глубокого анализа больших объемов данных о работе оборудования, эффективности процессов обогащения и качестве получаемой продукции. Современные технологии, включая машинное обучение и

искусственный интеллект, могут помочь в выявлении неэффективных звеньев производства, предсказании отказов оборудования и оптимизации использования ресурсов.

Существенное преимущество интеллектуальных систем заключается в их способности адаптироваться к меняющимся условиям производства и предоставлять оперативные рекомендации по корректировке процессов. Это достигается за счет непрерывного сбора и анализа данных, что позволяет системе «обучаться» и совершенствоваться в процессе эксплуатации. Таким образом, интеллектуальные системы становятся не просто инструментом управления, но и полноценным участником производственного процесса, способным предвидеть его исходы и влиять на них [17].

Внедрение интеллектуальных систем управления на ГОКах требует серьезной подготовки, включая аудит существующих процессов, подготовку технических решений и обучение персонала. Однако потенциальные выгоды от их использования – повышение производительности, снижение затрат и уменьшение воздействия на окружающую среду – делают эту задачу важной для будущего развития горно-обогатительной отрасли.

1.3. Обзор аналогов

В контексте российского рынка горнодобывающих технологий существует несколько значимых разработок, специализирующихся на создании систем управления для ГОКов. Эти системы направлены на повышение эффективности производства, безопасности и экологичности добычи и обогащения руд.

Интеллектуальная система «КАРАТ» – это комплексная автоматизированная система, разработанная для управления процессами добычи и транспортировки руды. Система «КАРАТ» включает в себя мониторинг состояния оборудования, управление грузопотоками и оптимизацию графиков работы. Система обеспечивает сбор и анализ данных в реальном

времени, что позволяет оперативно принимать решения по управлению производственными ресурсами.

VIST Group [12] предлагает решения для автоматизации горнодобывающих предприятий, включая системы планирования, диспетчеризации и управления технологическими процессами. Одним из ключевых продуктов является система управления карьерным транспортом VG KARIER, которая позволяет повысить эффективность использования автотранспорта и оборудования, снизить затраты на топливо и уменьшить время простоя.

MICROMINE – международная компания с сильным присутствием в России, предлагает комплексные решения для геологоразведки и горнодобывающей индустрии. Ее продукты, такие как Pitram и Micromine, предоставляют мощные инструменты для управления данными месторождений, планирования горных работ и оптимизации производственных процессов.

Российские разработчики предлагают ряд эффективных решений для автоматизации и управления процессами на горно-обогатительных комплексах [13-16]. Эти системы ориентированы на повышение производственной эффективности, снижение затрат и повышение уровня безопасности производственных процессов. Внедрение таких систем позволяет горнодобывающим компаниям не только оптимизировать свою деятельность, но и улучшить экологические показатели и соответствовать современным требованиям промышленной безопасности.

Вывод по первой главе

В этой главе проводится анализ горно-обогатительного комплекса и процессов обогащения руды, подчеркивая важность использования современных информационных технологий для повышения эффективности и экологической безопасности производства. В главе также обсуждается актуальность разработки и внедрения интеллектуальных систем управления, которые могут значительно улучшить оперативное управление и аналитику, способствуя более рациональному использованию ресурсов и минимизации воздействия на окружающую среду.

2. ТРЕБОВАНИЯ К СИСТЕМЕ

2.1. Функциональные и нефункциональные требования

Функциональные требования – это описание функций, которые должны выполнять программное обеспечение или система. Они определяют, какие действия или операции должны выполняться в ответ на определенные входные данные или события.

Разработанная система должна отвечать следующим требованиям:

1) система должна позволять выбирать желаемый период дат для отображения графиков, в том числе не только за конкретные даты, а также за кварталы, месяц и сутки;

2) система должна предоставлять сброс периода дат до базовых значений (период за сутки от текущего времени);

3) система должна предоставлять выбор трех вариантов рабочих смен: «все», «1 смена», «2 смена»;

4) система должна предоставлять меню включения и отключения графиков с помощью чекбоксов;

5) система должна предоставлять сворачивание и разворачивание меню, для полноценного просмотра графиков на всю страницу и выбора графиков соответственно;

6) система должна предоставлять просмотр графиков по выбранным чекбоксам;

7) система должна предоставлять скачивание графика в png формате.

Нефункциональные требования – это требования, которые определяют качественные характеристики системы, ее надежность, производительность, удобство использования и безопасность. Они не описывают конкретные функции системы, а обеспечивают ее качество и эффективность работы в целом. Для разработанной системы мониторинга на ГОКе можно выделить следующие ключевые нефункциональные требования.

1. Система должна быстро реагировать на действия пользователя, обеспечивая мгновенное отображение графиков и обновление данных без

значительных задержек. Загрузка данных и перерисовка графиков должны происходить в течение нескольких секунд, даже при больших объемах данных.

2. Система должна быть устойчивой к сбоям и гарантировать точность предоставляемой информации. Ошибки при обработке данных должны корректно обрабатываться, не приводя к аварийному завершению работы системы.

3. Система должна легко масштабироваться для поддержки увеличения объемов данных и роста числа пользователей без снижения производительности.

4. Интерфейс системы должен быть интуитивно понятным и удобным для пользователей без глубоких технических знаний. Навигация по функционалу системы должна быть логичной и простой.

5. Система должна корректно работать в основных современных браузерах и быть адаптирована для различных устройств.

6. Должна быть обеспечена возможность легкого обновления системы и ее компонентов.

2.2. Диаграмма вариантов использования

На основе анализа требований была разработана диаграмма вариантов использования. Диаграмма вариантов использования (use case diagram) (рисунок 1) подразумевает собой графическое представление функциональности системы, которая отображает, как разные типы пользователей могут взаимодействовать с системой.

Диаграмма вариантов использования состоит из актеров (actors) и вариантов использования (use cases). Актер представляет тип пользователя, который может взаимодействовать с системой, а варианты использования – это функциональные возможности системы, которые могут быть использованы пользователями.

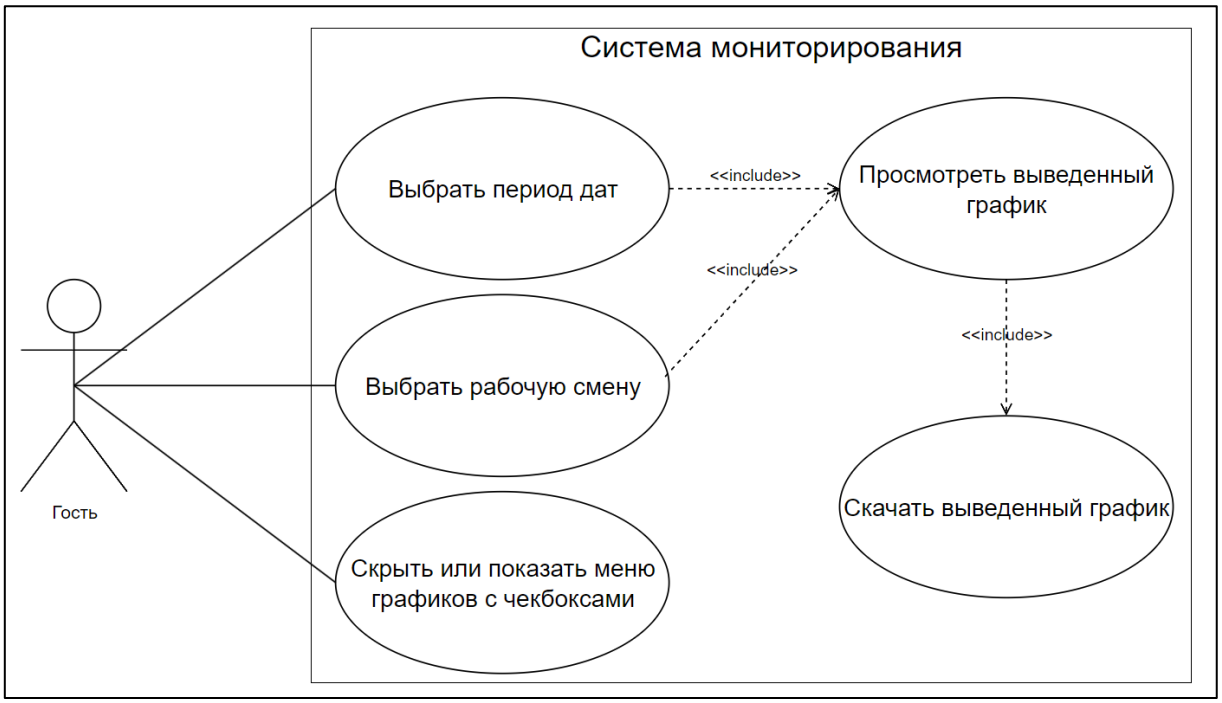

Рисунок 1 – Диаграмма вариантов использования

Система мониторинга АСУ ТП обогащения руды ГОКа является общедоступной для всех сотрудников корпоративной сети.

Вывод по второй главе

В этой главе определены ключевые функциональные и нефункциональные требования для мониторинга интеллектуальной системы управления горно-обогатительного комплекса. Описанные требования обеспечивают глубокое понимание необходимых функций и качественных характеристик системы, подчеркивают важность ее производительности, надежности и масштабируемости. Диаграмма вариантов использования помогает визуализировать взаимодействие пользователей с системой, упрощая последующие этапы проектирования и разработки.

3. АРХИТЕКТУРА СИСТЕМЫ

В этой главе представлен обзор технологического стека, применяемого в проекте, а также обоснование выбора конкретных технологий. Также описывается механизм взаимодействия между клиентской и серверной частями: детализируется процесс получения данных клиентским интерфейсом и его коммуникация с сервером.

3.1. Взаимодействие фронтенда и бэкенда

Диаграмма, изображенная на рисунке 2, представляет собой схему взаимодействия пользователя, фронтенда, бэкенда и базы данных в рамках системы, разработанной для мониторинга и отображения данных графика.

Схема демонстрирует, как данные проходят через различные этапы обработки от начального запроса пользователя до финальной визуализации графика, иллюстрируя взаимодействие между клиентским интерфейсом, серверной логикой и базой данных.

Далее приведено пошаговое описание процесса, изображенного на схеме.

1. Процесс начинается с действия пользователя, который нажимает на чекбокс для открытия графика. Это действие инициирует запросы в системе.

2. Клиентская часть системы (фронтенд), разработанная на базе React, принимает ввод от пользователя и формирует HTTP запрос, который включает параметры, такие как период дат, смена и ID графика. Этот запрос направляется на сервер.

3. Бэкенд сервер, работающий на FastAPI, получает запрос от клиентской части. Он обрабатывает запрос, обращаясь к базе данных PostgreSQL для извлечения нужных данных.

4. База данных PostgreSQL обрабатывает запрос сервера, извлекает требуемые данные и отправляет их обратно на сервер.

5. После получения данных от БД сервер преобразует эти данные в формат JSON и отправляет их обратно клиентской части.

6. Клиентская часть (фронтенд) получает данные в формате JSON. Затем происходит конвертация данных в формат, совместимый с библиотекой визуализации Apache Echarts.

7. После конвертации данных фронтенд производит визуализацию графика, представляя обработанные данные в графической форме для пользователя.

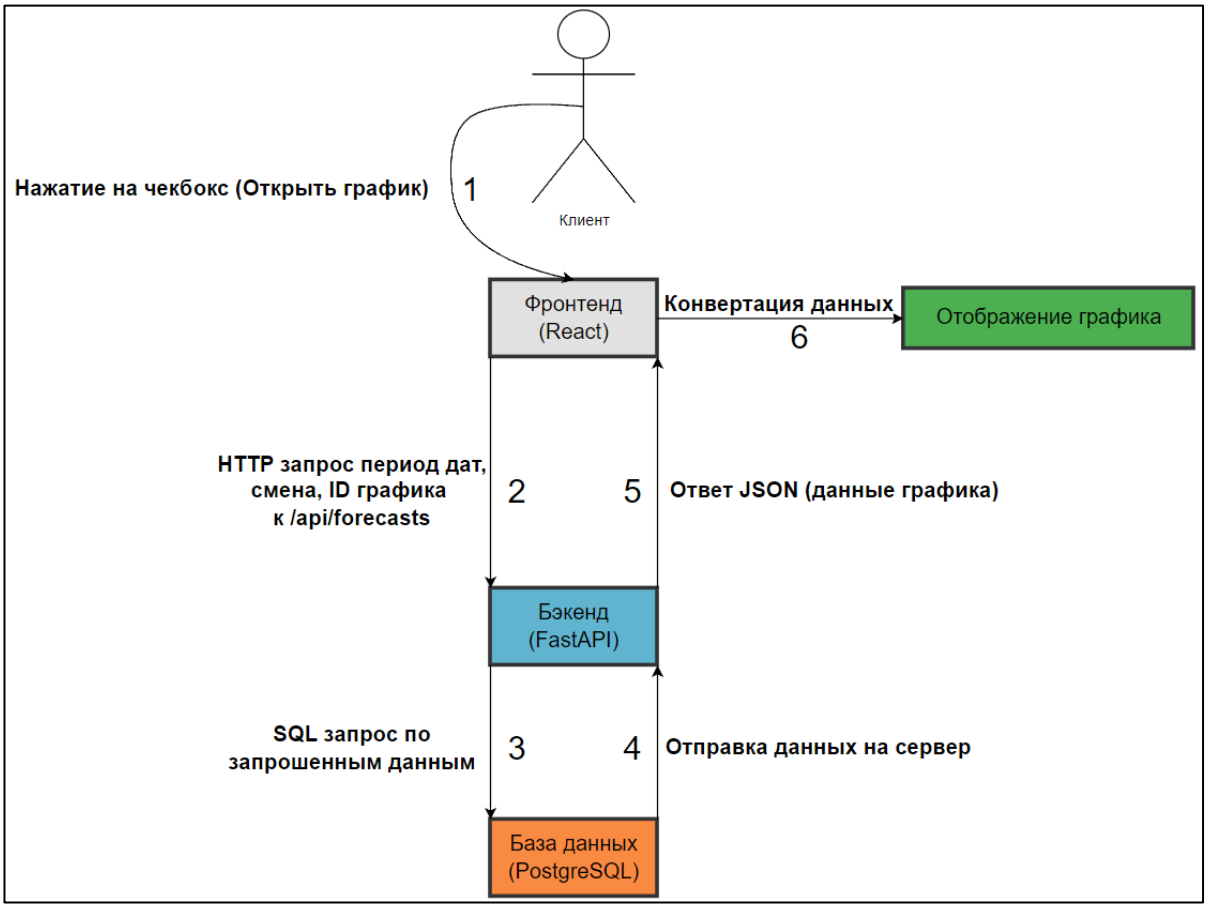

Рисунок 2 – Взаимодействие бэкенда и фронтенда

Когда пользователь переходит на страницу мониторинга (рисунок 3) он имеет возможность установить следующие параметры.

1. Пользователь может указать определенный временной диапазон и выбрать, за какой период дат и смену он хочет увидеть данные. Это может быть утренняя, дневная или вечерняя смена.

2. На странице реализованы разные графики, каждый из которых отображает определенный тип данных. Пользователь выбирает нужный график, нажимая на чекбокс рядом с его названием.

После выбора графика система автоматически отправляет запрос на бэкенд сервер, который управляется фреймворком FastAPI. Этот запрос включает в себя все параметры, которые пользователь установил: даты, смену и идентификатор выбранного графика.

Рисунок 3 – Представление веб-интерфейса

3.2. Стек используемых технологий при разработке системы

TypeScript

TypeScript [7] – это надстройка над JavaScript, разработанная Microsoft для добавления статической типизации в язык. Основная цель TypeScript – улучшить разработку крупных приложений и облегчить работу в команде за счет строгой проверки типов на этапе компиляции. Особенности
TypeScript.

1. TypeScript имеет статическую типизацию и позволяет заранее определять типы переменных, функций и объектов. Это помогает предотвратить множество ошибок еще до запуска кода, обеспечивая более надежную и безопасную базу для разработки.

2. Любой существующий JavaScript код является валидным и совместим с TypeScript кодом. TypeScript расширяет JavaScript, добавляя типизацию и некоторые другие возможности, но в итоге компилируется обратно в чистый JavaScript, обеспечивая широкую совместимость.

3. Статическая типизация позволяет инструментам разработки предоставлять лучшую подсказку кода, автодополнение и рефакторинг, делая процесс разработки более эффективным и менее подверженным ошибкам.

4. TypeScript опционален и не заставляет использовать типизацию везде и всегда. Можно постепенно добавлять типы в свои JavaScript проекты.

TypeScript был выбран для разработки данного проекта прежде всего из-за его возможностей статической типизации, что является значительным улучшением по сравнению с динамической типизацией JavaScript. Статическая типизация в TypeScript позволяет разработчикам определять типы переменных, функций и объектов на этапе написания кода, что значительно снижает вероятность возникновения ошибок на этапе выполнения программы. Это особенно важно в крупных проектах, где ошибки в типах данных могут привести к серьезным сбоям в работе приложения.

Кроме того, TypeScript обеспечивает отличную совместимость с JavaScript, поскольку любой JavaScript код может быть использован в TypeScript без изменений. Это делает TypeScript идеальным выбором для проектов, которые предполагают масштабирование и могут включать в себя как новый код, так и существующие JavaScript библиотеки.

React

React [6] – это библиотека для языка программирования JavaScript (TypeScript) с открытым исходным кодом для разработки пользовательских интерфейсов. Она помогает быстро и легко реализовать реактивность – явление, когда в ответ на изменение одного элемента меняется все остальное.

React используют для создания одностраничных и многостраничных приложений, разработки крупных сайтов. Например, с его помощью написано множество популярных сервисов, таких как, Twitter, TikTok, Pinterest, PayPal, AliExpress и Deepl, а также стриминговых сервисов, таких как, Netflix, Spotify и Twitch. Также на React реализованы новостные ленты крупнейших социальных сетей. Эта библиотека предназначена для:

1) создания функциональных интерактивных веб-интерфейсов, работая с которыми, не нужно постоянно обновлять страницу;

2) быстрой и удобной реализации отдельных компонентов и страниц целиком – элементы в React легко использовать повторно;

3) легкой разработки сложных программных структур – их просто описывать, если использовать реализованный в React подход;

4) доработки новой функциональности с любым изначальным стеком технологий: библиотека не зависит от остального инструментария и будет хорошо работать, на чем бы ни был написан код;

5) работы с серверной частью сайта или разработки мобильных приложений, также React используют совместно с инструментами, адаптирующими веб-технологии под другие цели.

Effector

Effector [5] – это эффективная и гибкая библиотека для управления состоянием в JavaScript и TypeScript приложениях, включая те, что построены с использованием React. Она предлагает простой и мощный API для работы со сложными состояниями и их мутациями, упрощая разработку реактивных приложений.

Далее приведены ключевые особенности Effector.

1. Сторы (Stores) – это основные единицы хранения данных и их состояния в Effector. Сторы реактивны и могут быть подписаны на изменения, что позволяет легко синхронизировать состояние с интерфейсом пользователя. Например, при открытом графике, можно изменять период дат и происходит новый запрос данных и на графике отрисовываются дан-

ные за вновь выбранный период. Это происходит благодаря подпискам на стор.

2. Эвенты (Events) – это события, которые могут изменять состояние сторов. Эвенты могут быть вызваны из пользовательского интерфейса или из эффектов для обозначения действий пользователя или системы.

3. Эффекты (Effects) – это асинхронные операции, такие как работа с API или другие побочные эффекты. Эффекты могут быть вызваны эвентами и могут изменять сторы, обеспечивая централизованное управление асинхронной логикой.

Apache Echarts

Apache Echarts [8] – это мощная, гибкая и высокопроизводительная библиотека для визуализации данных, доступная для использования в проектах на JavaScript и TypeScript. Echarts предоставляет широкий спектр типов графиков и диаграмм, что делает его удобным инструментом для представления сложных наборов данных в легко воспринимаемой визуальной форме. Библиотека разработана с особенным вниманием к деталям дизайна и интерактивности, позволяя пользователям не просто видеть данные, но и взаимодействовать с ними.

Далее приведены основные характеристики Apache Echarts.

1. Библиотека имеет широкий выбор типов диаграмм и поддерживает все, от простых линейных графиков до сложных тепловых карт и диаграмм Sankey. Это обеспечивает гибкость в выборе наиболее подходящего способа визуализации данных.

2. Каждый аспект графиков в Echarts может быть настроен, начиная от цветовой схемы и заканчивая поведением взаимодействия, таким как масштабирование и перетаскивание. Это позволяет создавать уникальные и полностью адаптированные визуализации.

3. Графики, созданные с помощью Echarts, не статичны. Они могут включать интерактивные элементы, такие как подсказки, легенды, которые

пользователи могут выбирать и настраивать, а также возможности масштабирования и перетаскивания.

4. Библиотека оптимизирована для работы с большими объемами данных, обеспечивая высокую производительность визуализаций даже при использовании сложных и объемных датасетов.

Использование Apache Echarts в проекте позволяет разработчикам трансформировать сырые данные в наглядные и понятные графики и диаграммы, значительно улучшая восприятие и анализ данных пользователями. Интеграция Echarts с React может быть выполнена через использование специальной обертки echarts-for-react, которая позволяет легко встраивать графики Echarts в React-компоненты.

FastAPI

FastAPI [1] – это высокопроизводительный фреймворк. Он использует Starlette для обработки запросов, что делает его одним из самых быстрых веб-фреймворков для Python, доступных на сегодняшний день. Это особенно ценно для приложений, требующих высокой производительности и масштабируемости. Фреймворк построен с использованием современных функций от Python 3.6, таких как асинхронность, которая позволяет выполнять асинхронные запросы и значительно улучшает эффективность ввода-вывода. Также он автоматически генерирует документацию для API с помощью Swagger UI или ReDoc, что облегчает тестирование API и делает его более доступным для фронтенд и бэкенд разработчиков.

Благодаря строгой типизации и автоматической валидации запросов и ответов, используя Pydantic, разработчики могут сократить количество ошибок и ускорить процесс разработки. FastAPI позволяет легко использовать асинхронные запросы к базам данных, асинхронную обработку файлов и другие асинхронные библиотеки, что делает его идеальным для высоконагруженных приложений.

PostgreSQL

PostgreSQL [3] – это бесплатная СУБД с открытым исходным кодом. С помощью PostgreSQL можно создавать, хранить базы данных и работать с данными с помощью запросов на языке SQL. Она поддерживает расширенные SQL-стандарты и функции, включая сложные запросы, подзапросы, триггеры, оконные функции и сохраненные процедуры, что делает эту СУБД идеальным выбором для сложных приложений, требующих глубокой обработки данных.

PostgreSQL высоко надежен, поддерживает большие объемы данных и масштабируется как вертикально, так и горизонтально, что обеспечивает гибкость при развитии проекта. База данных поддерживает создание и использование пользовательских функций, написанных на различных языках программирования, таких как Python, Java и даже Perl. Это позволяет адаптировать базу данных под конкретные нужды проекта. Эта СУБД предлагает мощные функции безопасности, включая поддержку SSL для шифрования соединений, сильную систему разграничения доступа и возможность расширения за счет сторонних решений для повышения безопасности данных.

Вывод по третьей главе

В этой главе представлен детальный обзор выбранных технологий и архитектурных решений для разработки системы мониторинга и визуализации данных. Обсуждаются причины выбора каждой технологии с акцентом на их функциональность, производительность и взаимодействие между клиентской и серверной частями.

4. РЕАЛИЗАЦИЯ

В данной главе представлена реализация ранее описанной архитектуры по разработке фронтенда и детальное описание отдельных процессов логики работы системы.

4.1. Формирование компонента меню мониторинга

Представление и описание работы меню в виде блок схемы

На представленной блок-схеме (рисунок 4) иллюстрируется процесс загрузки и отображения данных о меню мониторинга в веб-интерфейсе, используя фронтенд на базе React. Далее разобран каждый шаг схемы.

1. Пользователь заходит на веб-интерфейс, что инициирует взаимодействие с фронтенд-системой.

2. Фронтенд-система, построенная на React, активируется, и начинает процесс загрузки данных для меню мониторинга.

3. Функция запроса «сырых» данных о меню мониторинга «`fetchCheckboxDataFx`», используемая для отправки запроса к серверу (API) для получения данных, которые составляют меню мониторинга. Эта функция является частью системы управления состоянием приложения Effector, и она асинхронно получает данные от сервера.

4. Функция «`convertTreeDataToTreeFormat`» предназначена для преобразования данных, полученных по запросу. Она принимает исходные данные и преобразует их в структуру, подходящую для компонента Tree, который используется в React для отображения данных в виде иерархического дерева.

5. В компоненте Tree данные, уже преобразованные в нужный формат, загружаются для отображения в пользовательском интерфейсе. Компонент Tree представляет собой визуальный элемент, который позволяет пользователю взаимодействовать с данными, представленными в виде древовидной структуры.

6. Финальный этап, где меню мониторинга, составленное из элементов, представленных в компоненте Tree, отображается пользователю в веб-интерфейсе. Это позволяет пользователю взаимодействовать с меню, например, выбирать определенные параметры для отображения данных или графиков.

Блок-схема описывает полный процесс от момента входа пользователя на сайт до фактического отображения данных в интерфейсе, демонстрируя взаимодействие между клиентскими и серверными процессами в рамках веб-приложения.

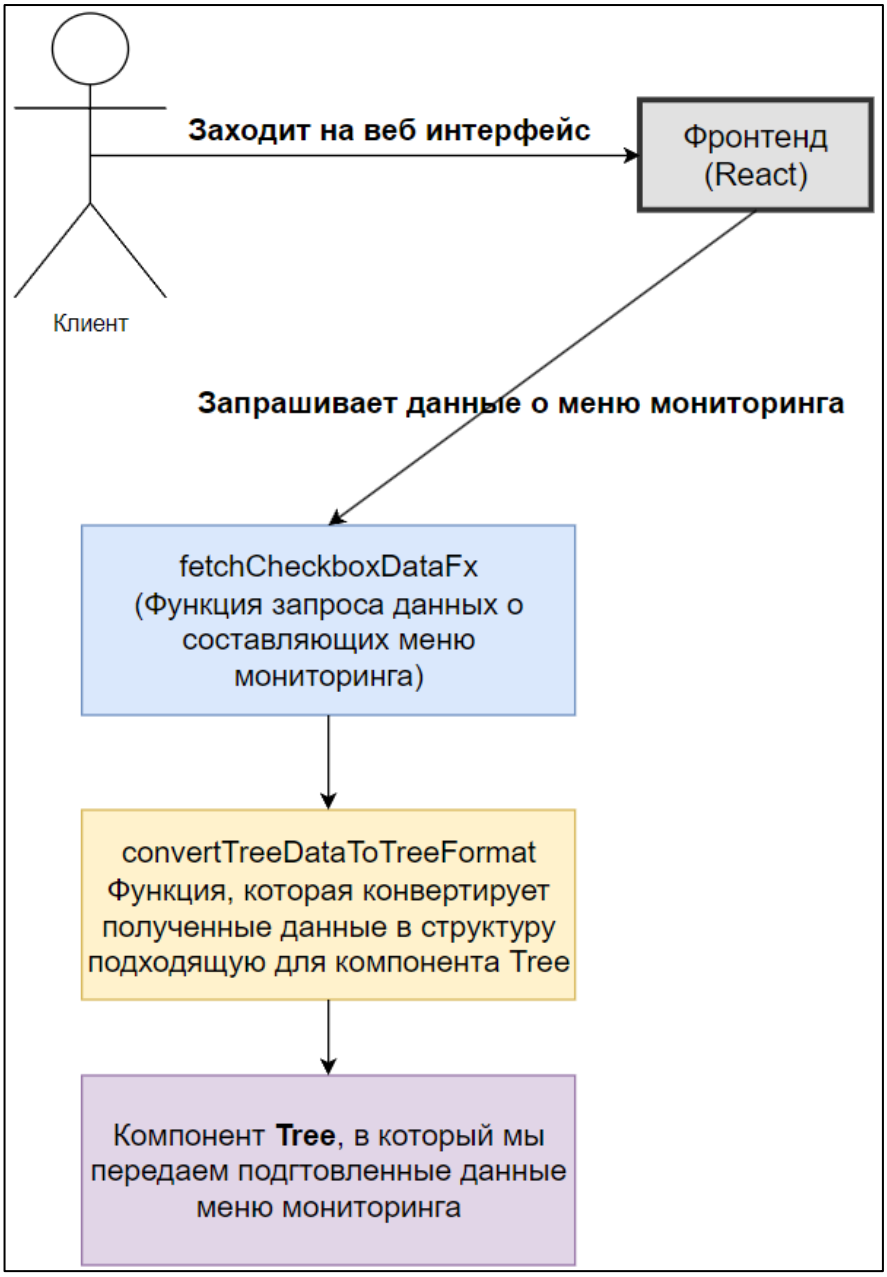

Рисунок 4 – Формирование компонента меню

Диаграмма взаимодействия компонентов

На диаграмме взаимодействия компонентов (рисунок 5) показан процесс обработки данных и взаимодействия между различными компонентами приложения. Далее разберем каждый компонент.

1. Компонент «`convertTreeItems`» подготавливает запрошенные данные о дереве мониторинга, преобразовывая исходный JSON в формат, пригодный для использования в компоненте дерева.

2. Компонент «`RenderCheckbox`» отвечает за отрисовку чекбоксов и реализацию логики и управлению их работы в интерфйесе.

3. Компонент «`SideBarMenu`» использует отрисованные чекбоксы и управляет их отображением в боковом меню.

4. Компонент «`PrepareChartData`» подготавливает "сырые" данные, полученные с сервера, в данные для использования в Echarts.

5. Компонент «`CollapsePanel`» оборачивает подготовленные данные в компонент сворачивающей панели.

6. Компонент «`ReactEcharts`» отвечает за финальную отрисовку графиков, используя подготовленные данные и отображая графики.

Диаграмма показывает, как данные преобразуются и проходят через различные этапы обработки, начиная с получения и преобразования данных до их финальной визуализации.

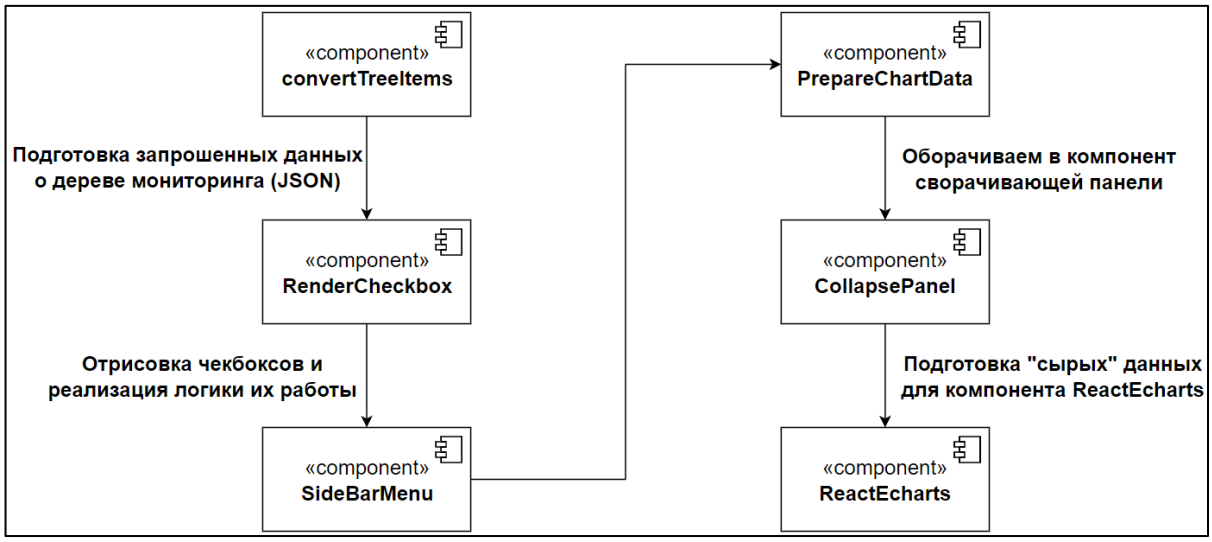

Рисунок 5 – Диаграмма взаимодействия компонентов

Логика работы меню мониторинга

Для начала нужно запросить данные для меню мониторинга «fetchCheckboxDataFx» (листинг 1) с помощью метода CreateEffect, который асинхронно запрашивает данные для меню мониторинга. Данные запрашиваются автоматически при первом заходе на страницу с помощью кастомного хука.

Листинг 1 – Функция запроса данных о меню мониторинга

```
export const fetchCheckboxDataFx = createEffect(async () => {
	const response = await fetch(
		`${window.appConfig.VITE_API_URL}scada_monitoring/menu`,
	);
	const jsonData = await response.json();
	return convertTreeDataToTreeFormat(jsonData);
});
```

Этот эффект выполняет HTTP-запрос к определенному URL, получает ответ в формате JSON (рисунок 6), и затем преобразует данные в нужный формат через функцию «convertTreeDataToTreeFormat» (листинг 2).

Листинг 2 – Функция адаптации данных для компонента

```
	import { MonitoringMenuTypeBackend }
from'../../Models/monitoring/effects/types';
	import { TreeItemNode } from '../../Models/monitoring/types';
	const convertTreeDataToTreeFormat = (
		origin: MonitoringMenuTypeBackend[],
	): TreeItemNode[] => {
		return origin.map((item) => ({
			id: String(item.id),
			title: item.title,
			indicatorId: item.indicator_id != null ?
String(item.indicator_id) : null,
			children: item.children ?
convertTreeDataToTreeFormat(item.children) : [],
		}));
	};
	export default convertTreeDataToTreeFormat;
```

Как видно из листинга 2, функция принимает данные, полученные с бэкенда, и возвращает массив адаптированных данных, подходящий для компонента дерева меню мониторинга.

Рисунок 6 – Часть полученного JSON по запросу для меню мониторинга

Структура данных JSON, которые получаем в ответ от сервера, включает в себе 4 поля.

1. Уникальный идентификатор (id) каждого объекта. Нужен для указания в параметр «key» в React, чтобы понимать какую часть интерфейса обновлять отдельно.

2. Уникальный идентификатор для каждого чекбокса (indicator_id) нужен для подготовки запроса данных из таблиц для бэкенда.

3. Название для выбранного показателя (title) в раскрывающейся панели отображения графика.

4. Если поле children имеет пустой массив, значит у него нет уникального идентификатора и оно является одним из родителей иерархии дерева, то есть у него не будет чекбокса. Если массив не пуст и у объектов есть indicator_id, то для этих элементов отрисовывается чекбокс.

4.2. Отрисовка чекбоксов к меню мониторинга

На текущем этапе меню мониторинга будет отображаться, но на показателях не будет чекбоксов. Чтобы можно было включать графики, нам нужно отобразить чекбоксы, а также написать логику работы для них. Чекбоксы должны не просто включаться и выключаться, но также хранить в себе состояние "включен" или "выключен". Помимо этого, чекбоксы должны содержать индикатор, который будет отправляться на сервер в строке запроса, отображать соответствующий график и инициировать запрос данных на сервер.

После запроса данных с помощью функции «fetchCheckboxDataFx», нужно создать еще одно хранилище «$checkboxData» (листинг 3) и подписаться на него с помощью метода «on» для CreateEffect из библиотеки Effector. Подписка ставится на «fetchCheckboxDataFx.doneData», то есть, когда данные будут полностью получены, будет активна подписка на этот запрос, и полученные данные запишутся в хранилище «$checkboxData».

Листинг 3 – Переменная (стор) для хранения данных о индикаторах меню

```
export const $checkboxData = createStore<TreeItemNode[]>([]).on(
      fetchCheckboxDataFx.doneData,
      (_, data: TreeItemNode[]) => data,
);
```

Для отрисовки чекбоксов реализован компонент «RenderCheckbox» (листинг 4), изображенный в виде блок-схемы на рисунке 7. В этом компоненте добавлена логика работы чекбокса. Компонент возвращает готовый блочный элемент с чекбоксами и логикой состояния. Когда по чекбоксу нажали, он включается или выключается, при этом отключается на время запроса данных. Также в эвент передаются данные id и title при нажатии. Этот компонент обеспечивает корректное взаимодействие с сервером, отображение графиков и управление состоянием чекбоксов, что делает его ключевым элементом в меню мониторинга.

Листинг 4 – Функция отрисовки чекбоксов

```
      import { Checkbox } from '@evraz/ui-kit';
      import clsx from 'clsx';
      import { useStore } from 'effector-react';
      import        {        toggleIndicatorSelection        }        from
'../../../Models/monitoring/events';
      import             {$chartStore,$selectedIndicators,}             from
'../../../Models/monitoring/model';
      import { RenderCheckboxProps } from './types';
      import styles from './RenderCheckbox.module.css';
      const    RenderCheckbox    =    ({    item,    className,    style    }:
RenderCheckboxProps) => {
            const selectedIndicators = useStore($selectedIndicators);
            const chartStore = useStore($chartStore);
            const handleCheckboxChange = () => {
                  if (item.indicatorId != null) {
                        toggleIndicatorSelection({
                              id: Number(item.indicatorId),
                              title: item.title,
                        });
                  }
            };
            const isChecked = selectedIndicators.some(
                  (indicator) => indicator.id === Number(item.indicatorId),
            );
            return (
                  <div
                        className={clsx(styles.itemContainer, className)}
                        style={style}
                        key={item.id}
                  >
                        <div className={styles.checkboxContainer}>
                              {item.indicatorId != null && (
                                    <Checkbox
                                          checked={isChecked}
                                          onChange={handleCheckboxChange}

      className={clsx(styles.customCheckboxSize)}
                                          disabled={chartStore.isLoading}
                                    />
                              )}
                        </div>
                        <div  className={styles.titleContainer}>{item.title}
</div>
                  </div>
            );
      };
      export default RenderCheckbox;
```

Как видно из листинга 4, функция отрисовывает чекбокс к каждому элементу, у которого id не null. Функция также добавляет логику отображения графика по нажатию чекбокса, при этом пока идет загрузка графика, все чекбоксы не будут доступны для выбора.

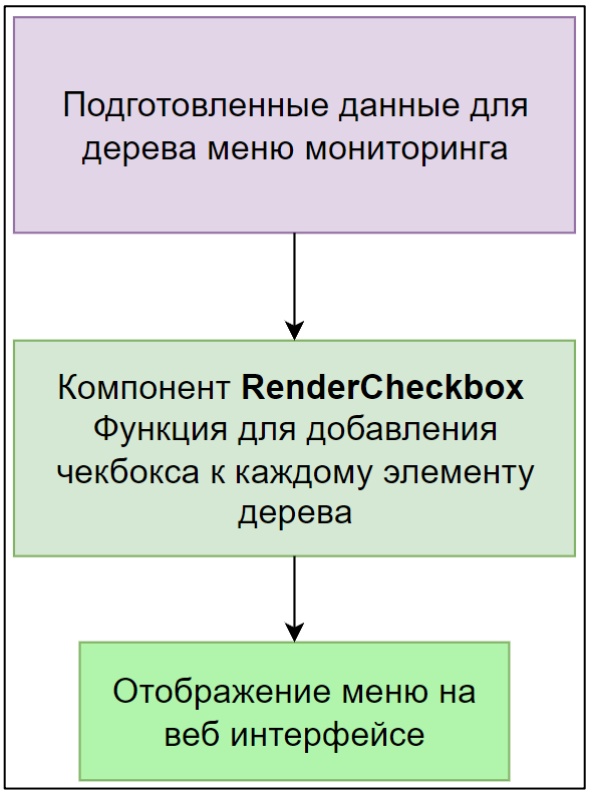

Рисунок 7 – Отрисовка чекбоксов к меню мониторинга

Далее используется вышеупомянутый компонент в компоненте «SideBarMonitoring» (листинг 5), в котором производится финальная отрисовка меню мониторинга. В данном компоненте используется кастомный хук «useCheckboxData» для запроса данных о структуре меню мониторинга. Здесь задействован «useEffect», который проверяет длину стора, и если стор не пустой, используется функция «fetchCheckboxDataFx» для запроса данных о меню мониторинга.

Затем подготовленные данные передаются в компонент «Tree», который отвечает за визуализацию всей структуры меню мониторинга (рисунок 8). Компонент «Tree» принимает данные о структуре меню и отображает их в виде дерева, где каждый узел представляет собой элемент меню с чекбоксом. Это позволяет пользователю наглядно видеть всю структуру меню и управлять отображением графиков с помощью чекбоксов.

Листинг 5 – Компонент отрисовки финального вида меню мониторинга

```
import { Tree } from '@evraz/ui-kit';
import clsx from 'clsx';
import { useStore } from 'effector-react';
import useCheckboxData from
'../../../Hooks/monitoring/useCheckboxData';
import { $sideBarData } from '../../../Models/monitoring/model';
import { TreeItemNode } from '../../../Models/monitoring/types';
import { BaseProps } from '../../../Shared/types';
import RenderCheckbox from '../RenderCheckbox';
import { TreeItemNodeWithRenderFn } from './types';
import styles from './SideBarMonitoring.module.css';

function SideBarMonitoring({ className, style }: BaseProps) {
      const sideBarData = useStore($sideBarData);

      useCheckboxData();

      const addRenderFnToItem = (item: TreeItemNode):
TreeItemNodeWithRenderFn => ({
            ...item,
            renderFn: () => <RenderCheckbox item={item} />,
            children: item.children &&
item.children.map(addRenderFnToItem),
      });

      const itemsWithRenderFn =
sideBarData.data.map(addRenderFnToItem);

      return (
            <div
                  className={clsx(className,
styles.main_page_sidebar_opened, {
                        [styles.main_page_sidebar_closed]:
!sideBarData.isSidebarOpen,
                  })}
                  style={style}
            >
                  <div>
                        <Tree items={itemsWithRenderFn}
allCollapsed={false} />
                  </div>
            </div>
      );
}
export default SideBarMonitoring;
```

Как видно из листинга 5 этот компонент позволяет динамически рендерить боковую панель с чекбоксами, которые пользователь может включать или выключать для управления отображаемым контентом.

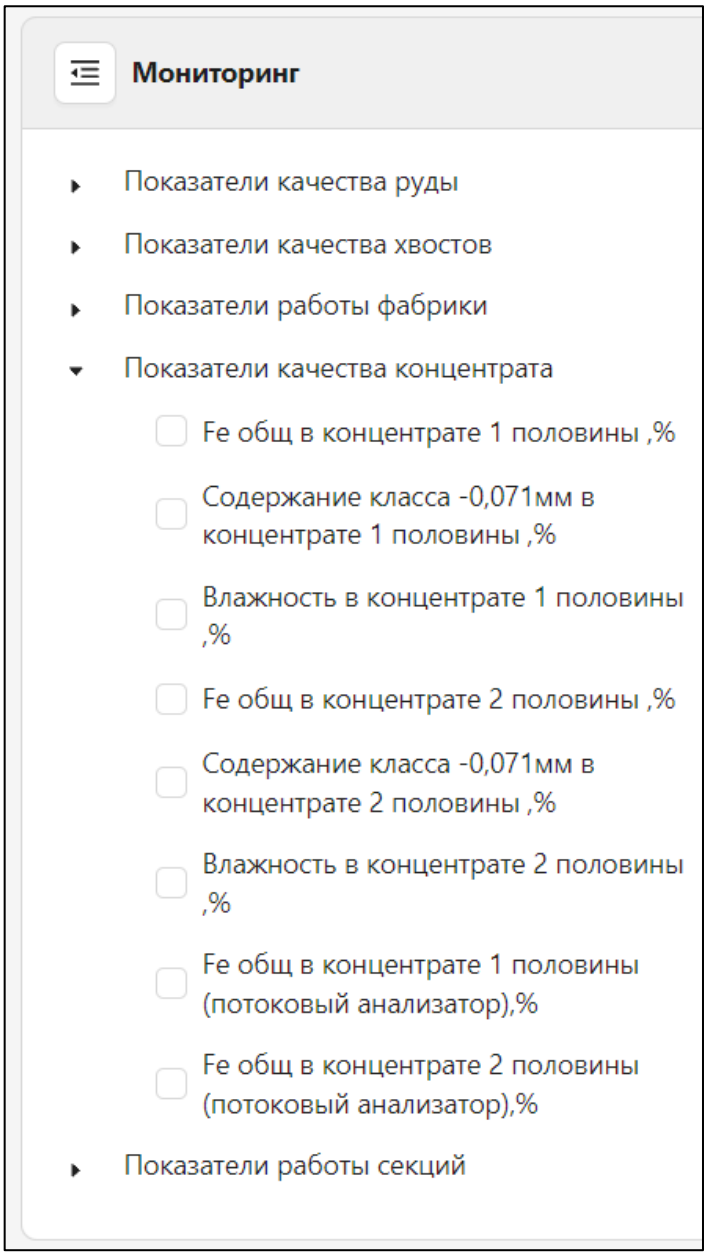

Рисунок 8 – Преобразованный JSON в готовом виде меню мониторинга

С помощью архитектурного подхода, основанного на использовании одного готового компонента Tree осуществляется контроль логики работы системы. Все остальные компоненты были реализованы в ходе выполнения ВКР, что позволяет в любой момент масштабировать приложение, удалять, добавлять или изменять логику чекбоксов.

Это позволяет адаптировать полученные данные под структуру меню мониторинга. Это простая функция подготовки данных дерева для компонента Tree, который будет отрисовывать само меню мониторинга.

4.3. Формирование строки запроса данных о графике

Прежде чем график появится на странице после нажатия чекбокса, нам нужно понять, как запрашиваются данные, такие как, период дат и/или рабочая смена. При выборе диапазона дат и/или смены компонента календаря «MonitoringFiltersLeft» они записываются в разные хранилища (Store) из библиотеки Effector и хранятся в этих сторах. Присутствуют такие хранилища как «$dateMin», «$dateMax», «$shift» (листинг 6). При первом клике по дате хранилище «$dateMin» записывает в себя минимальную дату, а хранилище «$dateMax» максимальную по второму клику на дату. Хранилище $shift хранит в себе выбранную рабочую смену. Стартовые значения для «$dateMin» это «oneDayAgo», что представляет собой текущую дату с вычетом 24 часов, для «$dateMax» это «currentDate», что представляет собой текущую дату. У хранилища «$shift» стартовое значение 0, что равносильно «Без смены», то есть отображать данные за весь период, также есть изменяемые параметры 1 или 2, «1 смена» и «2 смена» соответственно.

Листинг 6 – Хранилища (сторы) для периода дат и смен

```
export const $dateMin = createStore<Date>(oneDayAgo).on(
      updateDateMin,(_, date) => {return date},
);
export const $dateMax = createStore<Date>(currentDate).on(
      updateDateMax,(_, date) => {return date},
);
export const $shift = createStore('0').on(updateShift, (_, newShift) => {
      return newShift;
});
```

В приложении реализовано 4 основных параметра (таблица 1) запроса, таких как, indicator_id, date_min, date_max и shift. Параметр indicator_id берется изменю мониторинга, где у каждого из чекбоксов есть свой indicator_id. Из этих параметров формируется строка запроса. Когда выбирается период дат и/или смена, данные записываются в эти параметры и потом формируется строка для запроса к серверу вида «/api/monitoring?indicator_id=

772&date_min=20_01_2024T16:31:55&date_max=21_01_2024 T16:31:55& shift=0». Логика формирования изображена на рисунке 9.

Таблица 1 – Описание параметров запроса

Параметр запроса	Тип значения	Описание
indicator_id	number	Индикатор, соответствующий чекбоксу в меню мониторинга
date_min	Date	Начало периода даты
date_max	Date	Конец периода даты
shift	number	Выбор рабочей смены

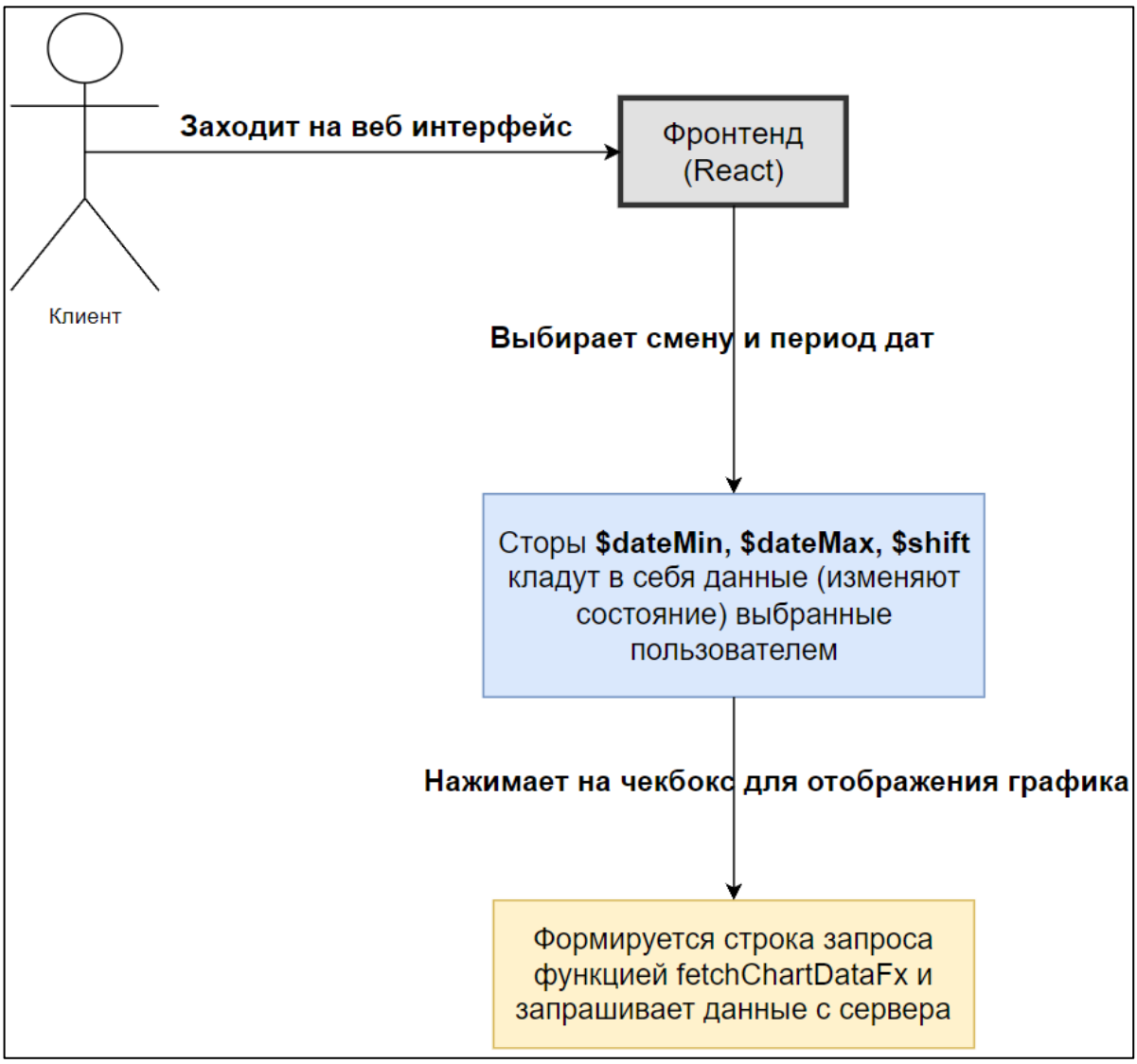

Рисунок 9 – Формирование строки запроса

4.4. Получение и обработка данных сервера после запроса

После того как выбираются даты и/или смену можно нажать на чекбокс в меню мониторинга для отображения графика. Каждый чекбокс соответствует определенному индикатору, который запрашивается.

Для запроса данных по выбранным индикаторам (чекбоксам) из меню мониторинга была реализована функция запроса данных «`fetchChartDataFx`» (листинг 7), которая также является хранилищем. В отличии от функции запроса данных о меню мониторинга, теперь нужно указать параметры для запроса данных «`queryParams`». Туда как раз и входят параметры, с которые упоминаются в пункте 4.2. После того как была отправлена сформированная строку запроса, веб-интерфейс получает данные от сервера.

Листинг 7 – Хранилища (сторы) для периода дат и смен

```
export const fetchChartDataFx = createEffect(
      async (parameters: FetchChartDataParams) => {
            const queryParams = prepareChartDataQueryParams(parameters);
            const response = await fetch(
      `${window.appConfig.VITE_API_URL}scada_monitoring/points?${queryParam
s}`,);
            const chartsData = await response.json();
            return  prepareChartData(chartsData,  parameters.indicatorIds);
      },
);
```

Как видно из листинга 7 функция запрашивает данные о графиках, выбранные пользователем, которые передаются через «`queryParams`».

Разберем структуру части полученных данных (листинг 8):

1) indicator_id – уникальный идентификатор для графика о который запрашивается и получаем данные;

2) name – название для сворачиваемой панели графика;

3) fact, recommendation, setpoint – данные вида массивов data_x и data_y, в которых есть timestamp и число, благодаря которым можно будет отобразить точку на графике.

Листинг 8 – JSON, приходящий с бэкенда

```
      "803": {"diapason": ["2024-05-19 00:00:00","2024-05-19 01:59:00"],
            "name": ["Fe общ в концентрате 1 половины ,%","Показатели
качества концентрата"],
            "fact":                                                {"data_x":
[1716064680.0,1716061200.0,1716058800.0],"data_y": [61.5,61.4,62.1]},
            "forecast": {
                  "data_x": [1716065940.0,1716069540.0],
                  "data_y": [61.873,61.558],
                  "color": [4,4]
            },
            "limits": {
                  "data_x": [1716065940.0,1716058800.0],
                  "data_y": [[61.1,61.7],[61.1,61.7]]}}
```

Данные пришедшие с бэкенда в формате JSON (рисунок 10) нужно привести к определенной структуре, подходящей для библиотеки Echarts, с помощью функции «prepareChartData», а конкретно объекту «convertToSeriesData» (листинг 9). В ней приводятся данные в определенную структуру, подходящую для библиотеки Echarts, и создаем так называемые серии данных для каждой линии из пришедших данных с бэкенда.

Листинг 9 – Функция конвертации данных графика

```
const convertToSeriesData = (data: DataSeries): SeriesDataPoint[] => {
            return data.data_x.map((timestamp, index): SeriesDataPoint => {
                  const value = data.data_y[index];
                  return { value: [timestamp * 1000, value] };
            });
            };
```

Как видно из листинга 8 функция конвертирует полученные данные, полученные с бэкенда в массив, подходящий для библиотеки Apache Echarts. Умножение на 1000 нужно, потому что TypeScript считает время в миллисекундах, а с бэкенда данные времени приходят в секундах.

В этом объекте принимаются на вход сырые данные, которые прислал на сервер и проходимся методом map и сопоставляем поочередно значение из data_x и data_y. На выходе получаем массив вида [1710440700, 61.4], где первое значение – это время в формате timestamp по оси X, а второе значение по оси Y. Умножение timestamp на 1000 объясняется тем, что

JavaScript (TypeScript) считает время в миллисекундах, а данные с сервера приходят в секундах.

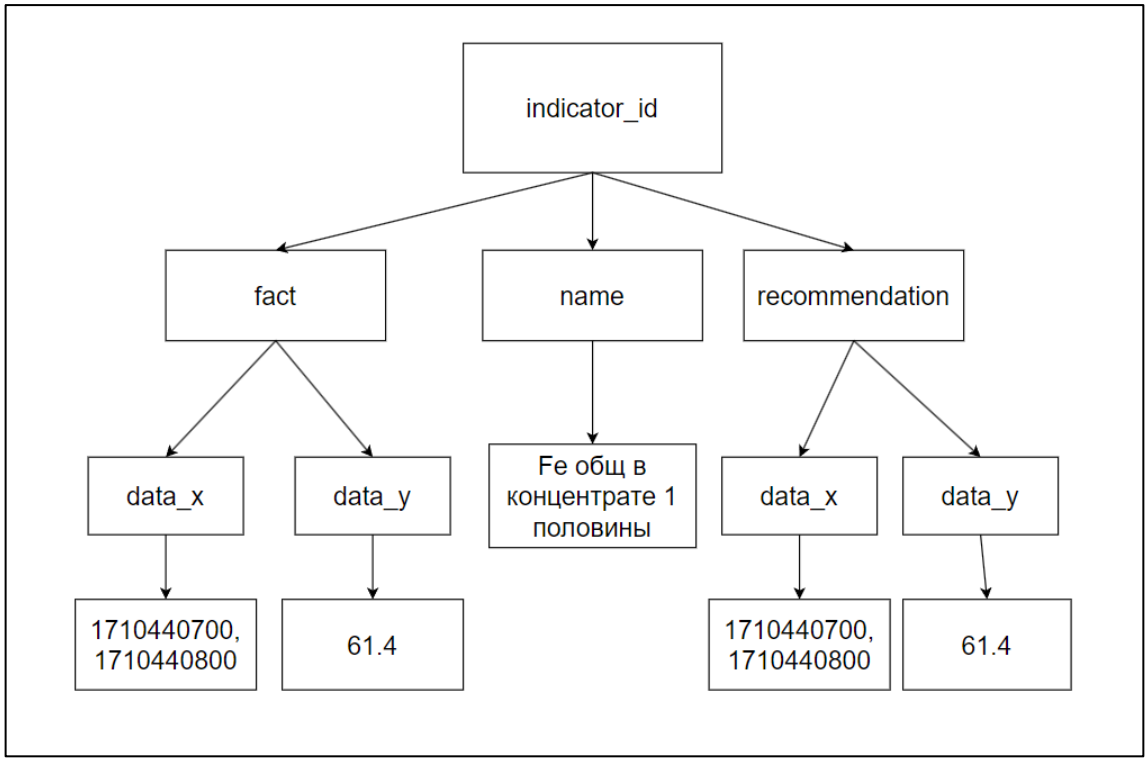

Рисунок 10 – JSON данные графика в древовидной структуре

Разберем одну из серий данных. Сначала идет проверка, есть ли значения для этой серии. Далее добавляется в массив объект, где name это отображаемое имя линии или точки, type – тип линии, может быть точка, сплошная или прерывистая линия и многие другие значения, color это цвет линии в различных вариантах: hex, rgb, rbga и многие другие, в data данные преобразуются в определенной структуры массив через функцию «`convertToSeriesData`», в которую передаются данные фактической линии.

Серий на графике может быть разное количество, от одной до пяти. Обычно это либо одна, либо две, либо пять серий для каждого графика. Для одной серии обычно отображается линия факта, которая отображается фактическое значение технологической секции. Если серий две, то обычно это серии факта и рекомендации. Если серий пять, то это серии факта, рекомендации, уставки, лимитов и режим работы секций. Каждая серия име-

ет свой характер отображения, серия факта, уставки и лимитов отрисовывается как линия, серия рекомендаций отрисовывается как точки, серия режима работы секций отрисовывается как фоновый цвет графика, на каждом из определенных промежутков свой цвет фона.

4.5. Вывод подготовленных данных в компонент Echart

После того как сырые данные преобразованы в данные, подходящие для библиотеки Echarts, можно перейти к реализации итогового отображения графиков и их точной настройке через компонент Echart. Используем хранилище «`$selectedIndicators`», добавляем несколько условий для того, чтобы пока данные запрашивались, отображался значок загрузки. Далее используем метод map по подготовленным данным и заворачиваем каждый график в компонент «`CollapsePanel`», для того чтобы график имел видимые поля и самое главное, чтобы его можно было сворачивать и разворачивать. В этот компонент передаются стили css и title это название для компонента, который отображается на светло-фиолетовом фоне. После этого добавляем несколько дополнительных проверок на существование данных в этом хранилище и используем компонент «`ReactECharts`» из библиотеки Echarts для отображение самого графика. В этом компоненте есть один параметр options.

Разберем подробнее что передается в этот параметр. Параметр grid это отступы графика в пикселях, для того чтобы центровать график в случае, если длинные даты или значения по осям. Параметр «`xAxis`» это настройки оси X, в нее передается параметр type со значением time, чтобы обозначить какой будет тип оси X, также через параметр «`formatdate`» форматируем timestamp в дату вида «dd.mm.yyyy HH:mm». В «`yAxis`» передаем тип value, так как по оси Y выводятся числа. В параметр series передается подготовленный массив данных, который разобран в предыдущем пункте. Отображаем tooltip при наведении на линию. Также добавляем параметр toolbox с определенными надстройками для того, чтобы выведен-

ный график можно было скачать, нажав по соответствующей иконке. Блок-схема логики работы изображена на рисунке 11.

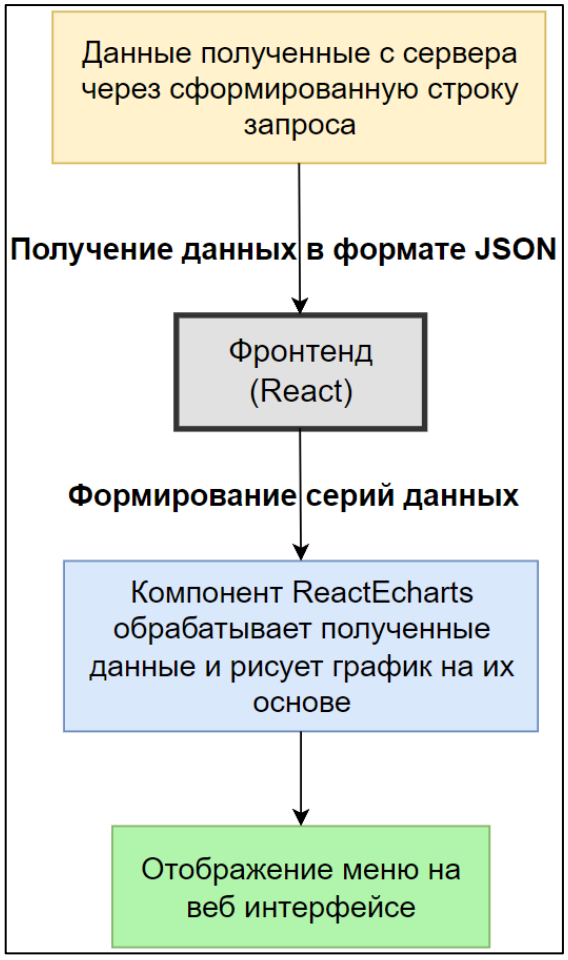

Рисунок 11 – Логика отображения данных в график

Вывод по четвертой главе

В данной главе описывается практическая сторона внедрения архитектуры системы, начиная от фронтенд-компонентов до обработки и отображения данных на стороне клиента. Рассматриваются ключевые этапы создания пользовательского интерфейса, включая запросы данных, их обработку и визуализацию в виде интерактивных графиков. Описанная реализация подчеркивает эффективное взаимодействие между фронтендом и бэкендом.

5. ТЕСТИРОВАНИЕ

5.1. Методология тестирования

Тестирование фронтенд-части интеллектуальной системы автоматизированного управления технологическими процессами (САУТП) обогащения руды для ГОКа проводилось с целью обеспечения высокой надежности и качества работы приложения [13]. Основное внимание уделялось тестированию адаптивности и кросс-браузерной совместимости, а также проверке функциональности основных компонентов системы.

Для достижения этих целей были выбраны следующие направления тестирования:

1) адаптивное тестирование, чтобы убедиться в корректном отображении и работе приложения на устройствах с различными разрешениями экрана;

2) кросс-браузерное тестирование, направленное на проверку работоспособности приложения в различных веб-браузерах;

3) функциональное тестирование, фокусирующееся на проверке правильности выполнения основных функций приложения.

5.2. Процесс тестирования

Процесс тестирования начался с проверки адаптивности приложения. Это включало в себя использование различных устройств, таких как мобильные телефоны, планшеты и настольные компьютеры, чтобы убедиться, что пользовательский интерфейс корректно отображается и функционирует на всех типах экранов. Особое внимание было уделено тому, чтобы элементы интерфейса, такие как кнопки, меню и графики, корректно масштабировались и оставались удобными для использования независимо от размера экрана. Тестирование проводилось через инструменты разработчика (devtools) в браузерах, чтобы эмулировать различные разрешения экрана, такие как 1920x1080, 1792x828 (iPhone XR), и 768x1024 (iPad).

Далее было проведено кросс-браузерное тестирование. Приложение тестировалось в наиболее популярных веб-браузерах, включая Google Chrome, Mozilla Firefox, Microsoft Edge и Safari. Это было необходимо для обеспечения того, чтобы все элементы интерфейса корректно отображались и функционировали одинаково хорошо во всех браузерах. В процессе тестирования проверялись различные сценарии использования, чтобы гарантировать отсутствие ошибок и сбоев. Было также проведено тестирование на разных операционных системах, включая Windows, macOS и iOS.

Функциональное тестирование заключалось в проверке основных функций приложения. Это включало тестирование выбора периода дат, выбора смены, отображения и скачивания графиков. Каждая из этих функций была тщательно проверена на соответствие требованиям спецификации и корректность работы. Было важно убедиться, что пользовательский интерфейс интуитивно понятен и удобен в использовании, что значительно улучшает пользовательский опыт.

5.3. Результаты тестирования

Результаты тестирования показали, что приложение успешно справляется с задачами, поставленными перед ним [17]. В ходе адаптивного тестирования было подтверждено, что интерфейс корректно отображается и функционирует на всех тестируемых устройствах. Элементы интерфейса масштабировались корректно, и пользователи могли легко взаимодействовать с приложением независимо от размера экрана. Тестирование включало проверку на таких устройствах, как iPhone XR, iPad, а также на настольных компьютерах с различными разрешениями экрана.

Кросс-браузерное тестирование показало, что приложение работает стабильно во всех тестируемых браузерах. Все элементы интерфейса корректно отображались и функционировали, обеспечивая единообразный пользовательский опыт [16]. Это подтвердило, что приложение готово к

использованию пользователями, независимо от их предпочтений в выборе браузера.

Функциональное тестирование подтвердило, что все основные функции приложения работают корректно. Пользователи могли выбирать периоды дат, смены и отображать соответствующие графики без каких-либо проблем. Кроме того, возможность скачивания графиков в формате PNG также работала исправно, что позволяет пользователям сохранять важные данные для последующего анализа.

Вывод по пятой главе

Тестирование подтвердило, что разработанная фронтенд-часть интеллектуальной системы автоматизированного управления технологическими процессами обогащения руды для ГОКа соответствует всем заданным требованиям [20]. Адаптивное тестирование показало, что приложение корректно работает на устройствах с различными разрешениями экрана, кросс-браузерное тестирование подтвердило стабильную работу во всех популярных веб-браузерах, а функциональное тестирование продемонстрировало корректность выполнения основных функций.

Все проведенные тесты были успешно пройдены, что подтверждает готовность системы к эксплуатации. Полученные результаты свидетельствуют о высоком качестве и надежности разработанного приложения, обеспечивающего эффективное и удобное управление процессами обогащения руды на горно-обогатительном комплексе.

ЗАКЛЮЧЕНИЕ

В рамках данной работы было разработано фронтенд приложение мониторирования и контроля процессов ГОКа. При этом были решены следующие задачи:

1) спроектирована архитектура и макет системы мониторинга;
2) добавлены функции для запроса данных с эндпоинтов;
3) разработаны функциональные компоненты выбора дат и смен;
4) разработано меню мониторинга;
5) реализованы и настроены графики библиотеки Echarts;
6) протестирована система мониторинга.

В будущем планируется добавление страниц «Главная» и «Архивные данные» для расширения приложения и удобства отслеживания и контроля операторами процессов ГОКа, а также написание своей библиотеки компонентов с расширенной логикой и настроек стилей.

Разработанная система будет внедряться на предприятие и являться немаловажным инструментом в руках персонала ГОКа, предоставляющая контроль над производственным процессом, улучшение экологичности производства и безопасности.

ЛИТЕРАТУРА

1. Создание API | FastAPI. [Электронный ресурс] URL: https://fastapi.tiangolo.com/tutorial/ (дата обращения: 07.01.2024 г.).

2. SQLAlchemy: The Database Toolkit for Python. [Электронный ресурс] URL: https://docs.sqlalchemy.org/en/20/ (дата обращения: 11.01.2024 г.).

3. PostgreSQL. [Электронный ресурс] URL: https://www.postgresql.org/docs/ (дата обращения: 19.01.2024 г.).

4. Data validation library | Pydantic. [Электронный ресурс] URL: https://docs.pydantic.dev/latest/api/base_model/ (дата обращения: 22.01.2024 г.).

5. Глобальный стейт менеджер | Effector. [Электронный ресурс] URL: https://effector.dev/ru/ (дата обращения: 03.02.2024 г.).

6. JavaScript-библиотека | React. [Электронный ресурс] URL: https://ru.legacy.reactjs.org/ (дата обращения: 11.02.2024 г.).

7. Типизированный язык программирования (надстройка над JavaScript) | TypeScript. [Электронный ресурс] URL: https://www.typescriptlang.org/ (дата обращения: 17.02.2024 г.).

8. Библиотека визуализации | Apache Echarts. [Электронный ресурс] URL: https://echarts.apache.org/en/option.html#title (дата обращения: 13.02.2024 г.).

9. HTTP-клиент для запросов | Axios. [Электронный ресурс] URL: https://axios-http.com/ru/docs/intro (дата обращения: 13.02.2024 г.).

10. Автоматизация ГОК. [Электронный ресурс] URL: https://www.summatechnology.ru/projects/branch/mining/ (дата обращения: 13.02.2024 г.).

11. Горно-металлургическая компания | EVRAZ. [Электронный ресурс] URL: http://it.evraz.com/about (дата обращения: 11.02.2024 г.).

12. Разработчик цифровых решений на ГОК | ВИСТ Групп. [Электронный ресурс] URL: https://www.zyfra.com/ru/industries/mining/ (дата обращения: 22.01.2024 г.).

13. Пучков Л.А., Федунец Н.И., Потресов Д.К. Автоматизированные системы управления в горнодобывающей промышленности: Учебник для вузов. – М.: Недра, 1987, – 285 с.

14. Автоматизация горнодобывающей промышленности. [Электронный ресурс] URL: https://evomatics.ru/solutions/avtomatizatsiya-gornodobyvayushchey-promyshlennosti/ (дата обращения: 22.03.2024 г.).

15. Автоматизация горнодобывающих предприятий промышленности. [Электронный ресурс] URL: www.sinergo.ru/services/avtomatizaciya-gornodobyvayushchih-predpriyatij/ (дата обращения: 13.03.2024 г.).

16. Применение современных систем автоматизации на открытых горных работах промышленности. [Электронный ресурс] URL: https://cyberleninka.ru/article/n/primenenie-sovremennyh-sistem-avtomatizatsii-na-otkrytyh-gornyh-rabotah/viewer (дата обращения: 11.04.2024 г.).

17. Практика применения специализированного ПО в горной промышленности. [Электронный ресурс] URL: https://zolteh.ru/technic/praktika_primeneniya_spetsializirovannogo_po_v_gornoy_promyshlennosti/ (дата обращения: 17.03.2024 г.).

18. Автоматизированные системы контроля. [Электронный ресурс] URL: https://gaskar.group/ru/industries/mining-sector (дата обращения: 11.04.2024 г.).

19. Автоматизация горнодобывающей промышленности. [Электронный ресурс] URL: https://akinginiring.ru/catalog/avtomatizatsiya-gornodobyvayushchey-promyshlennosti/ (дата обращения: 14.04.2024 г.).

20. Информационные технологии в горном деле: Учеб. пособие/ Ю.Н. Попков, А.Ю. Прокопов, М.В. Прокопова/ Шахтинский ин-т (филиал) – Новочеркасск: ЮРГТУ, 2007. – 202 с.