МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования   
«САНКТ–ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой № 44

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| проф., д-р техн. наук, проф. |  |  |  | М.Б. Сергеев |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

БАКАЛАВРСКАЯ РАБОТА

|  |  |
| --- | --- |
| на тему | Разработка веб-интерфейса для отображения взаимодействия настроек на 3D |
| визуализацию цифрового телеуправляемого рентгеновского аппарата | |
|  | |

|  |  |
| --- | --- |
| выполнена | Никифоровым Никитой Сергеевичем |
| фамилия, имя, отчество студента в творительном падеже | |

|  |  |  |  |
| --- | --- | --- | --- |
| по направлению подготовки | 09.03.01 |  | Информатика и вычислительная |
|  | код |  | наименование направления |
| техника | | | |
| наименование направления | | | |
| направленности | 04 |  | Компьютерные технологии, системы и |
|  | код |  | наименование направленности |
| сети | | | |
| наименование направленности | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент группы № | 4142 |  | 10.06.2025 |  | Н.С. Никифоров |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Руководитель

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| канд. техн. наук, доцент |  |  |  | Н.Н. Решетникова |
| должность, уч. Степень, звание |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2025

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования   
«САНКТ–ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

УТВЕРЖДАЮ

Заведующий кафедрой №44

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| проф., д-р техн. наук, проф. |  |  |  | М.Б. Сергеев |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

ЗАДАНИЕ НА ВЫПОЛНЕНИЕ БАКАЛАВРСКОЙ РАБОТЫ

|  |  |  |  |
| --- | --- | --- | --- |
| студенту группы | 4142 |  | Никифоров Никита Сергеевич |
|  | номер |  | фамилия, имя, отчество |

|  |  |
| --- | --- |
| на тему | Разработка Web интерфейса для отображения взаимодействия настроек на 3D |
| визуализацию цифрового телеуправляемого рентгеновского аппарата | |
|  | |

|  |  |  |  |
| --- | --- | --- | --- |
| утвержденную приказом ГУАП от | 27.03.2025 | № | 11-387/25 |

|  |  |
| --- | --- |
| Цель работы: | Разработка Web интерфейса для отображения взаимодействия |
| настроек на 3D визуализацию цифрового телеуправляемого рентгеновского аппарата | |
|  | |

|  |  |
| --- | --- |
| Задачи, подлежащие решению: | Анализ предметной области разработки веб интерфейсов. |
| Выбор программных средств для разработки интерактивной 3D визуализации. Создание | |
| 3D модели телеуправляемого рентгеновского аппарата. Создание пользовательского | |
| интерфейса для взаимодействия с 3D моделью рентгеновского аппарата. Создание | |
| способа настройки возможностей взаимодействия с рентгеновским аппаратом. Создание базы данных для хранения данных 3D моделей. Создание сервера эмулятора рентгеновского аппарата. | |

|  |  |
| --- | --- |
| Содержание работы (основные разделы): | 1. Анализ предметной области. |
| 1. Создание 3D модели рентгеновского аппарата. 3) Отображение 3D модели в | |
| Web приложении. 4) Создание пользовательского интерфейса. 5) Создание сервера | |
| 6) Создание сервера эмулятора рентгеновского аппарата. 7) Проверка работоспособности Web приложения. | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Срок сдачи работы « | 10 | » | Июня | 20 | 25 |

Руководитель

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| канд. техн. наук, доцент |  |  |  | Н.Н. Решетникова |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

Задание принял(а) к исполнению

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| студент группы № | 4142 |  | 10.06.2025 |  | Н.С. Никифоров |

**Оглавление**

[ВВЕДЕНИЕ 7](#_Toc198204946)

[1. Анализ предметной области 9](#_Toc198204947)

[1.1. Описание предметной области 9](#_Toc198204948)

[1.2. Описание исходных данных для создания 3D-модели телеуправляемого рентгеновского аппарата 9](#_Toc198204949)

[1.3. Описание исходных данных для создания веб интерфейса 11](#_Toc198204950)

[1.4. Сравнение с существующими аналогами веб-интерфейсов для отображения настроек в реальном времени. 11](#_Toc198204951)

[1.5. Выбор программных средств 13](#_Toc198204952)

[1.6. Выводы по анализу предметной области 15](#_Toc198204953)

[2. Создание 3D модели рентгеновского аппарата 15](#_Toc198204954)

[3. Отображение 3D модели рентгеновского аппарата 15](#_Toc198204955)

[3.1. Общее описание процесса визуализации 15](#_Toc198204956)

[3.2. Настройка сцены и рендеринга 16](#_Toc198204957)

[3.3. Загрузка и отображение модели 16](#_Toc198204958)

[3.4. Оптимизация производительности 17](#_Toc198204959)

[3.5. Выводы 17](#_Toc198204960)

[4. Создание пользовательского интерфейса 17](#_Toc198204961)

[4.1. Общее описание интерфейса 17](#_Toc198204962)

[4.2. Структура интерфейса 18](#_Toc198204963)

[4.3. Интеграция с конфигурацией 18](#_Toc198204964)

[4.4. Реализация интерактивности 19](#_Toc198204965)

[4.5. Стилизация и адаптивность 19](#_Toc198204966)

[4.6. Выводы 19](#_Toc198204967)

[5. Создание сервера 20](#_Toc198204968)

[5.1. Архитектура серверной части 20](#_Toc198204969)

[5.2. Разработка REST API 21](#_Toc198204970)

[5.3. Реализация базы данных 24](#_Toc198204971)

[5.4. Интеграция с Socket.IO 26](#_Toc198204972)

[5.5. Выводы 27](#_Toc198204973)

[6. Создание сервера эмулятора рентгеновского аппарата. 28](#_Toc198204974)

[7. Проверка работоспособности приложения 29](#_Toc198204975)

[7.1. Цели и методы тестирования 29](#_Toc198204976)

[7.2. Тестирование отображения модели 29](#_Toc198204977)

[7.3. Тестирование пользовательского интерфейса 31](#_Toc198204978)

[7.4. Тестирование серверной части 32](#_Toc198204979)

[7.5. Тестирование базы данных 33](#_Toc198204980)

[7.6. Проверка производительности 33](#_Toc198204981)

[7.7. Выводы 33](#_Toc198204982)

[ЗАКЛЮЧЕНИЕ 34](#_Toc198204983)

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

**Blender** — программное обеспечение с открытым исходным кодом для 3D-моделирования, анимации и рендеринга, используемое для создания реалистичных трёхмерных объектов.

**Three.js** — JavaScript-библиотека, использующая WebGL для отображения и управления 3D-графикой в веб-браузерах.

**WebGL** — веб-стандарт для рендеринга 3D-графики в браузере без дополнительных плагинов.

**glTF (GL Transmission Format)** — формат файлов для эффективной передачи и загрузки 3D-моделей и сцен, разработанный Khronos Group.

**JSON (JavaScript Object Notation)** — текстовый формат обмена данными, используемый для конфигурации и хранения информации в приложениях.

**SQLite** — легковесная реляционная база данных, предназначенная для хранения данных с минимальными системными требованиями.

**bun.js** — серверная платформа для выполнения JavaScript-кода, используемая для создания серверных приложений.

**Express** — фреймворк для Node.js, упрощающий разработку веб-серверов и API.

**UI (User Interface)** — пользовательский интерфейс, обеспечивающий взаимодействие между пользователем и приложением.

**API (Application Programming Interface)** — программный интерфейс приложения, позволяющий взаимодействовать с внешними системами или компонентами.

# ВВЕДЕНИЕ

В современном мире цифровые технологии прочно интегрированы в нашу повседневную жизнь. Веб приложения становятся инструментом для решения задач в разных областях. Особое место занимают приложения с интерактивными 3D моделями, которые позволяют пользователям визуализировать сложные объекты и взаимодействовать с ними через сетевое соединение. Такие решения находят применение в медицине, образовании, промышленности и научных исследованиях, обеспечивая новые возможности для обучения, демонстрации и управления оборудованием. В связи с этим разработка веб-приложений с использованием 3D становится актуальной задачей.

Разработка веб-интерфейса для отображения взаимодействия настроек с 3D-визуализацией цифрового телеуправляемого рентгеновского аппарата представляет собой актуальную задачу в условиях роста спроса на дистанционные и интерактивные технологии. Телеуправляемые рентгеновские аппараты широко используются для диагностики, обеспечивая высокую безопасность за счёт дистанционного управления. Визуализация таких систем в веб-среде позволяет не только демонстрировать их конструкцию, но и моделировать работу, что особенно полезно для обучения медицинского персонала и презентации оборудования.

Цель данной дипломной работы заключается в разработке веб-интерфейса, который обеспечивает возможность интерактивной настройки 3D модели телеуправляемого рентгеновского аппарата. Для моделирования используется именно телеуправляемый аппарат так как другие модели не имеют дистанционного управления. Особенность данной работы состоит в создании реалистичной 3D-визуализации, интегрированной с пользовательским интерфейсом, а также в эмуляции работы аппарата через серверное решение. Это требует комплексного подхода, включающего моделирование, программирование и работу с данными.

В рамках дипломной работы проведен анализ современных технологий для 3D визуализации и веб-разработки, создана 3D модель рентгеновского аппарат с использованием Blender, реализован веб-интерфейс с использованием библиотеки Three.js, а также разработаны сервер эмулятор и база данных SQLite для хранения данных конфигураций. Проект направлен на решение таких задач как обучение операторов и демонстрации возможностей оборудования для повышения доступности сложных систем.

Результаты данной работы могут быть использованы в образовательных учреждениях для подготовки специалистов в области медицинской техники. А также в системах управления рентгеновскими аппаратами. Также эта разработка может способствовать развитию использования технологий 3D визуализации в веб-приложениях в смежных областях, и областях где может применятся сложная телеуправляемая техника.

Таким образом, данная дипломная работа направлена на создание инновационного решения, объединяющего передовые технологии и практическую значимость, а также на повышение интереса к изучению современных методов визуализации для управления сложными системами.

1. Анализ предметной области
   1. Описание предметной области

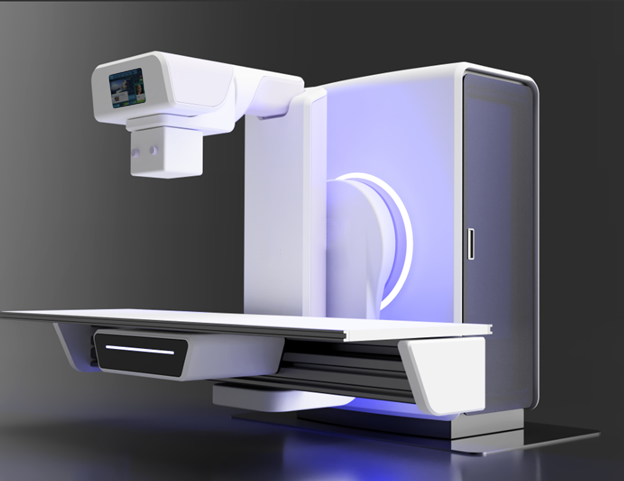
В современном мире, в условиях быстрого развития цифровых технологий веб-интерфейсы становятся неотъемлемой частью многих продуктов. 3D веб-визуализации становятся важным инструментом в различных областях таких как медицина, образование, научные исследования. Такие приложения позволяют пользователям визуализировать сложное оборудование и взаимодействовать с ним изменяя настройки и наблюдая за изменениями в реальном времени.

Предметная область включает в себя создание реалистичных 3D-моделей, их интеграцию в веб-среду, разработку интуитивно понятного пользовательского интерфейса и обеспечение хранения данных. Для успешной реализации проекта необходимо сочетание знаний в области 3D-графики, веб-разработки и особенностей работы медицинского оборудования, такого как телеуправляемые рентгеновские аппараты.

Работа в данном проекте сосредоточена на телеуправляемых рентгеновских аппаратах, поскольку именно они поддерживают удаленное управление, что делает их идеальными для интеграции с веб-интерфейсами. Такие аппараты позволяют оператору изменять настройки, такие как положение стола или угол наклона излучателя, дистанционно через программное обеспечение, обеспечивая высокую точность и безопасность.

* 1. Описание исходных данных для создания 3D-модели телеуправляемого рентгеновского аппарата

Для создания 3D-модели телеуправляемого рентгеновского аппарата в качестве исходных данных используются схемы и технические характеристики реальных устройств, применяемых в медицинской практике. Телеуправляемый рентгеновский аппарат представляет собой комплексное оборудование, включающее рентгеновский излучатель, детектор, подвижный стол для пациента и систему управления. Рендер планируемого к выпуску рентгеновского аппарата представлен на рисунке 1.2.

  
Рисунок 1.2 – Рендер предполагаемого к выпуску рентгеновского аппарата

Основные элементы, подлежащие моделированию, включают:

**Детектор**: фиксирует излучение, проходящее через объект, для формирования изображения.

**Подвижный стол**: позволяет изменять положение пациента относительно излучателя и детектора.

**Механизмы управления**: шарниры и приводы, обеспечивающие точное позиционирование.

Примерные размеры аппарата зависят от конкретной модели, но типичные параметры включают высоту около 2 м, ширину 1,5–2 м и длину стола 2–2,5 м. Технические характеристики, такие как диапазон углов наклона стола (от -90° до +90°).

* 1. Описание исходных данных для создания веб интерфейса

Для создания веб-интерфейса телеуправляемого рентгеновского аппарата в качестве исходных данных используются схемы пультов управления реальных устройств, применяемых в медицинской практике. Схема пульта управления представлена на рисунке 1.3.

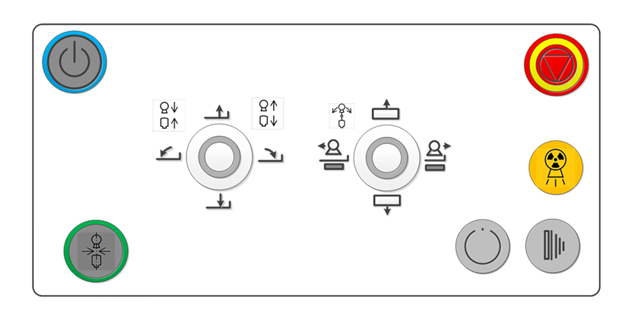


Рисунок 1.3 – Схема пульта управления

Для использования на кнопках предполагаются к использованию символы стандарта “ISO 7000 / IEC 60417 Graphical symbols for use on equipment”, это коллекция символов , которые могут быть размещены на оборудовании, чтобы указать, как правильно и безопасно его использовать. Она включает в себя символы для всех типов оборудования.

* 1. Сравнение с существующими аналогами веб-интерфейсов для отображения настроек в реальном времени.

На рынке не существует полных аналогов но из близких решений есть программа Voron для управления 3D принтером, она позволяет управлять им в реальном времени и просматривать создание 3D модели, отправленной на печать по слоям. На рисунках 1.4.1-1.4.2 представлен интерфейс данной программы.

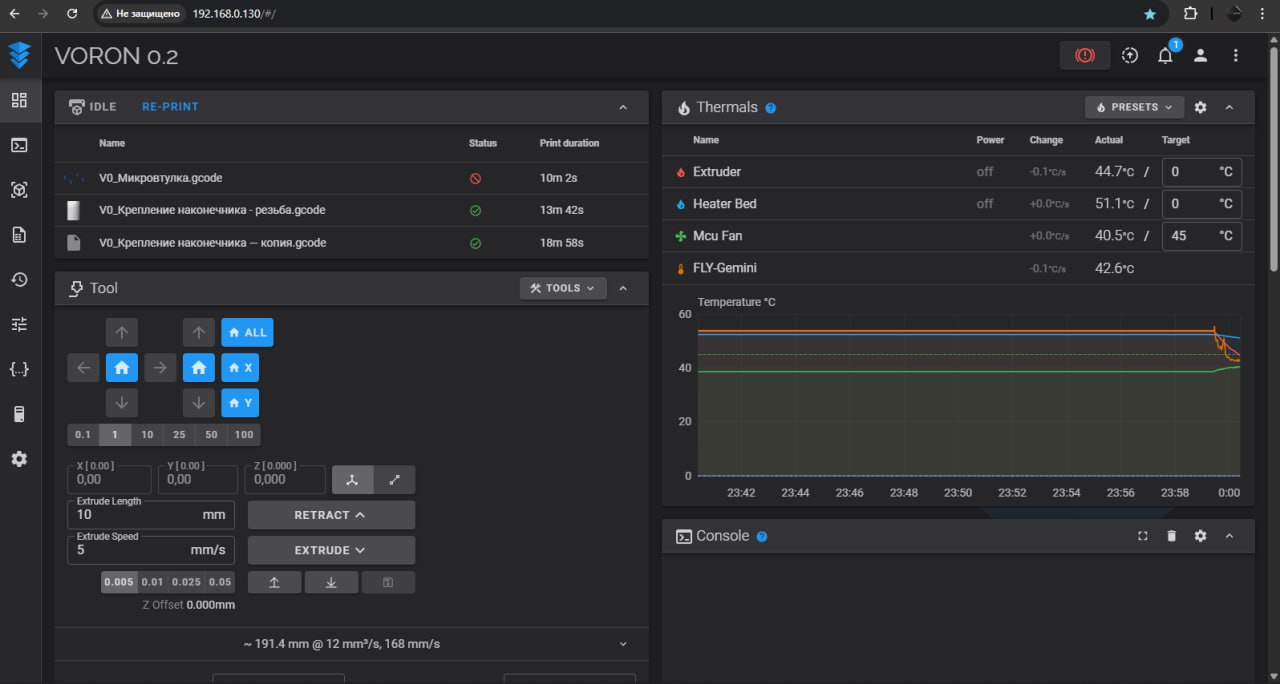


Рисунок 1.4.1 – Интерфейс управления 3D принтером

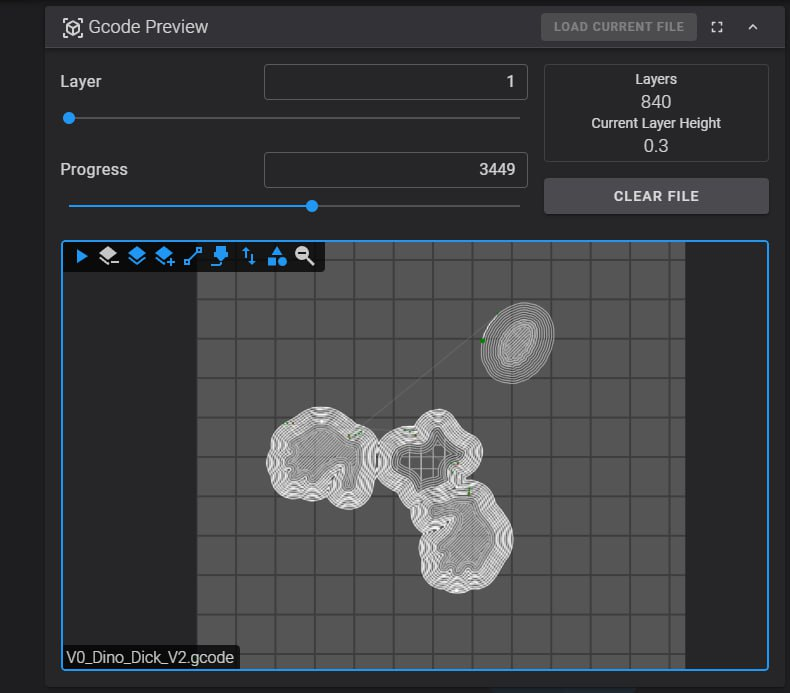


Рисунок 1.4.2 – Визуализация слоя печати

* 1. Выбор программных средств

Для реализации проекта были выбраны следующие инструменты и технологии, исходя из их функциональности, совместимости и соответствия поставленным задачам:

**1. Blender**

Blender был использован для создания реалистичных 3D моделей для последующей загрузки в программу.

**Особенности**:

* Поддержка полигонального моделирования и создания реалистичных текстур;
* Возможность экспорта моделей в формат glTF 2.0 (glb), совместимый с Three.js;
* Широкий набор инструментов для детализированного воспроизведения сложных механизмов, таких как рентгеновский аппарат;
* Бесплатность и активное сообщество разработчиков.

**Причины выбора**: Blender выбран за его универсальность, бесплатность и совместимость с Three.js. Альтернативы, такие как Autodesk Maya, были отклонены из-за высокой стоимости и избыточной функциональности для задач проекта.

**2. Three.js**

Three.js был использован для рендеринга 3D графики в браузере, является JavaScript-библиотекой, использующий WebGL для рендеринга

**Особенности**:

* Простота интеграции с веб-приложениями и поддержка формата glTF;
* Возможность создания интерактивных сцен с управлением камерой и объектами;
* Высокая производительность при работе с 3D-моделями в реальном времени;
* Совместимость с современными браузерами без дополнительных плагинов.

**Причины выбора**: Three.js выбран за простоту интеграции и высокую производительность. Альтернативы, такие как Babylon.js, были отклонены из-за более сложной настройки для задач проекта.

**3. JSON**

JSON используется для конфигурации возможностей взаимодействия с моделью.

**Особенности**:

* Лёгкость парсинга в JavaScript, что упрощает интеграцию с Three.js;
* Компактность и низкая вероятность ошибок при настройке параметров;
* Высокая читаемость и поддержка в JavaScript-экосистеме.

**Причины выбора**: JSON выбран за его простоту и нативную поддержку в JavaScript. Альтернатива YAML была рассмотрена, но отклонена из-за избыточной сложности для небольшого числа моделей.

**4. SQLite**

SQLite использовался для хранения 3D-моделей и конфигураций как легковесная реляционная база данных для хранения данных

**Особенности**:

* Отсутствие необходимости в отдельном серверном процессе, что снижает нагрузку на систему;
* Простота интеграции с веб-приложением благодаря компактности и встраиваемости;
* Достаточная производительность для небольшого объёма данных;
* Поддержка параметризованных запросов для безопасности.

**Причины выбора**: SQLite выбрана за её лёгкость, компактность и отсутствие необходимости в серверной инфраструктуре. Альтернатива MongoDB была отклонена из-за минимальной ожидаемой нагрузки и избыточных требований к ресурсам.

**5. Bun.js с Express**

Bun.js и фреймворк Express выбраны для создания сервера-эмулятора рентгеновского аппарата.

**Особенности**:

* Высокая совместимость с JavaScript-экосистемой, включая Three.js и JSON;
* Простота реализации REST API для обработки запросов от интерфейса;
* Быстрое развертывание и поддержка асинхронных операций;
* Встроенная поддержка SQLite, обеспечивающая высокую скорость запросов.

**Причины выбора**: Bun.js с Express выбраны за высокую производительность и простоту интеграции с SQLite. Альтернатива Node.js была отклонена из-за меньшей скорости работы для задач, для которых оптимизировано приложение.

**6. Socket.IO**

Socket.IO был использован для организации двунаправленного соединения в реальном времени между клиентом, сервером и эмулятором.

**Особенности**:

* Поддержка WebSocket с автоматическим переподключением, что обеспечивает стабильность соединения;
* Простота реализации событийной модели для передачи команд и состояний;
* Высокая совместимость с JavaScript-экосистемой, включая Bun.js и Express;
* Возможность рассылки обновлений всем подключённым клиентам через broadcast.

**Причины выбора**: Socket.IO выбран за простоту интеграции, поддержку реального времени и надёжность. Альтернативы, такие как WebSocket API или MQTT, были отклонены: WebSocket API требует больше кода для обработки переподключений, а MQTT избыточен для веб-приложений с небольшой нагрузкой.

* 1. Выводы по анализу предметной области

Анализ предметной области показал, что разработка веб-интерфейса для телеуправляемого рентгеновского аппарата требует использования современных технологий 3D-визуализации и веб-разработки. Выбор Blender, Three.js, JSON, SQLite и Bun.js, Socket.io обусловлен их функциональностью, производительностью и совместимостью. В отличие от существующих аналогов, проект фокусируется на интерактивности и эмуляции работы оборудования, что делает его уникальным решением для обучения и демонстрации.

1. Создание 3D модели рентгеновского аппарата
2. Отображение 3D модели рентгеновского аппарата
   1. Общее описание процесса визуализации

Отображение 3D-модели телеуправляемого рентгеновского аппарата в веб-приложении является ключевой частью проекта, обеспечивая визуальное представление устройства и его интерактивность. Для реализации визуализации использовалась библиотека Three.js, которая позволяет рендерить 3D-графику в браузере с использованием WebGL. Модель аппарата создавалась в Blender и экспортировалась в формате glTF 2.0 (glb), что обеспечивает компактность и высокую производительность при загрузке.

Основной задачей было обеспечить корректное отображение модели с реалистичным освещением, тенями и возможностью управления её положением и ориентацией. Для этого в проекте реализована сцена с настроенными источниками света, камерой и инструментами управления, такими как TrackballControls.

* 1. Настройка сцены и рендеринга

Настройка сцены реализована в файле src\additions\setupLCC.tsx, где создаются основные компоненты Three.js:

* **Сцена** (THREE.Scene) служит контейнером для всех объектов.
* **Камера** (THREE.PerspectiveCamera) настроена с углом обзора 75 градусов и позицией (20, 20, -5), что обеспечивает удобный обзор модели.
* **Рендерер** (THREE.WebGLRenderer) активирует поддержку теней (shadowMap.enabled = true) и использует сглаживание (antialias: true) для повышения качества изображения.
* **Освещение** включает окружающий свет (AmbientLight, интенсивность 1.6) для равномерного освещения и направленный свет (DirectionalLight, интенсивность 2.3) для создания теней. Параметры направленного света настроены для точного контроля области теней (shadow.camera).

Управление сценой осуществляется через TrackballControls, позволяющий пользователю вращать, масштабировать и перемещать модель. Параметры управления (rotateSpeed, zoomSpeed, panSpeed) оптимизированы для плавного взаимодействия.

* 1. Загрузка и отображение модели

Загрузка модели реализована с использованием GLTFLoader из Three.js, как показано в файле src\components\ModelViewer.tsx. Модель в формате glb запрашивается с сервера через API (/api/models/:modelName), после чего парсится и добавляется в сцену. Код в src\hooks\useModelLoader.ts отвечает за:

* Парсинг бинарных данных модели (loader.parse).
* Добавление модели в сцену (scene.add(gltf.scene)).
* Настройку теней для всех объектов типа Mesh (castShadow и receiveShadow).

Для инициализации позиций и поворотов компонентов модели используется структура данных ModelPositions, которая синхронизируется с конфигурацией из JSON-файла, загружаемого через API (/api/json/:modelName). Это позволяет динамически задавать начальные параметры модели и её интерактивных элементов.

* 1. Оптимизация производительности

Для обеспечения высокой производительности реализованы следующие меры:

* Использование формата glb минимизирует объём данных.
* Ограничение размеров теневых карт (shadow.mapSize) до 1024x1024 пикселей балансирует качество и производительность.
* Очистка сцены и ресурсов при размонтировании компонента (scene.clear, renderer.dispose) предотвращает утечки памяти.
  1. Выводы

Разработанный механизм отображения 3D-модели рентгеновского аппарата обеспечивает реалистичную визуализацию с поддержкой теней и интерактивного управления. Использование Three.js и glTF позволило достичь высокой производительности и совместимости с современными браузерами.

1. Создание пользовательского интерфейса
   1. Общее описание интерфейса

Пользовательский интерфейс (UI) разработан для обеспечения интуитивного взаимодействия с 3D-моделью рентгеновского аппарата. Основной задачей было предоставить пользователю возможность изменять настройки модели (положение, поворот) и управлять её компонентами (например, излучателем или столом) с использованием различных элементов управления. Интерфейс реализован на React с использованием библиотек Redux Toolkit для управления состоянием и React Router для навигации.

* 1. Структура интерфейса

Интерфейс включает следующие ключевые компоненты, реализованные в папке src\components:

* **Навигация** (Navigation.tsx): фиксированная панель с ссылками на главную страницу, страницу загрузки модели и авторизацию. Стилизована с использованием UnoCSS для адаптивного дизайна.
* **Просмотрщик модели** (ModelViewer.tsx): основной компонент, отображающий 3D-модель и элементы управления. Поддерживает динамическую загрузку моделей по имени через API.
* **Элементы управления**:
  + **Джойстик** (SControlJoystick.tsx): позволяет изменять положение или поворот компонентов модели по двум осям (x, y). Реализован с использованием библиотеки react-joystick-component.
  + **Кнопка установки** (SSetButton.tsx): активирует предустановленные значения для компонентов модели, обеспечивая плавное изменение параметров с шагом 0.1.
  + **Числовые поля ввода** (ModelControlsInputs.tsx): позволяют задавать точные значения положения и поворота для каждой оси (x, y, z) с учётом заданных ограничений.
* **Выбор модели** (SLSelect.tsx): выпадающий список для выбора доступных моделей, загружаемых через API (/api/models).
* **Загрузка модели** (AddModel.tsx): форма для добавления новых моделей с предварительным просмотром (ModelPreview.tsx).
  1. Интеграция с конфигурацией

Конфигурация элементов управления задаётся в JSON-файлах, которые загружаются с сервера (/api/json/:modelName). Структура данных, описанная в src\types\models.ts, включает:

* ModelControls: список моделей с параметрами позиционирования (position) и поворота (rotation) для каждой оси.
* controlElements: массив элементов управления (джойстик, кнопка), где указаны их свойства, такие как целевые компоненты и пути изменения значений (xpath, ypath).

Файл src\hooks\useModelData.ts отвечает за загрузку и парсинг JSON-конфигурации, которая затем передаётся в компоненты ModelControlsComponent и ModelControlsInputs для рендеринга интерфейса.

* 1. Реализация интерактивности

Интерактивность реализована через хук useModelLoader.ts, который:

* Синхронизирует изменения в ModelPositions с положением и поворотом объектов в сцене.
* Обрабатывает события от джойстика (движение в направлениях FORWARD, BACKWARD, LEFT, RIGHT) и кнопки установки, изменяя параметры с заданным шагом (0.1).
* Учитывает ограничения, заданные в конфигурации (xyzController), для предотвращения некорректных значений.

Для управления состоянием используется Redux Toolkit (src\store\model\model.api.ts), обеспечивающий асинхронные запросы к серверу и кэширование данных.

* 1. Стилизация и адаптивность

Стилизация выполнена с использованием UnoCSS и кастомных CSS-файлов (src\index.css, src\components\shared\css\). Основные особенности:

* Тёмная тема с акцентными цветами (#5863f8, #efe9f4) для визуального комфорта.
* Адаптивный дизайн с использованием flexbox для корректного отображения на разных устройствах.
* Кастомные стили для джойстика и кнопок, обеспечивающие единообразный вид.
  1. Выводы

Пользовательский интерфейс обеспечивает удобное и гибкое взаимодействие с 3D-моделью рентгеновского аппарата. Реализованные элементы управления (джойстик, кнопки, поля ввода) позволяют точно настраивать параметры модели, а интеграция с JSON-конфигурацией делает интерфейс масштабируемым.

1. Создание сервера

Серверная часть является ключевым компонентом системы, обеспечивающим взаимодействие между веб-интерфейсом, базой данных и эмулятором телеуправляемого рентгеновского аппарата. Бэкенд отвечает за обработку запросов на загрузку и управление 3D-моделями, настройку подключений к эмулятору и передачу данных в реальном времени. Для реализации использованы Bun.js как высокопроизводительная среда выполнения JavaScript, фреймворк Express для создания REST API и библиотека Socket.IO для двунаправленного взаимодействия. В данной главе описаны архитектура сервера, разработка API, интеграция с базой данных, реализация реального времени и результаты тестирования.

* 1. Архитектура серверной части

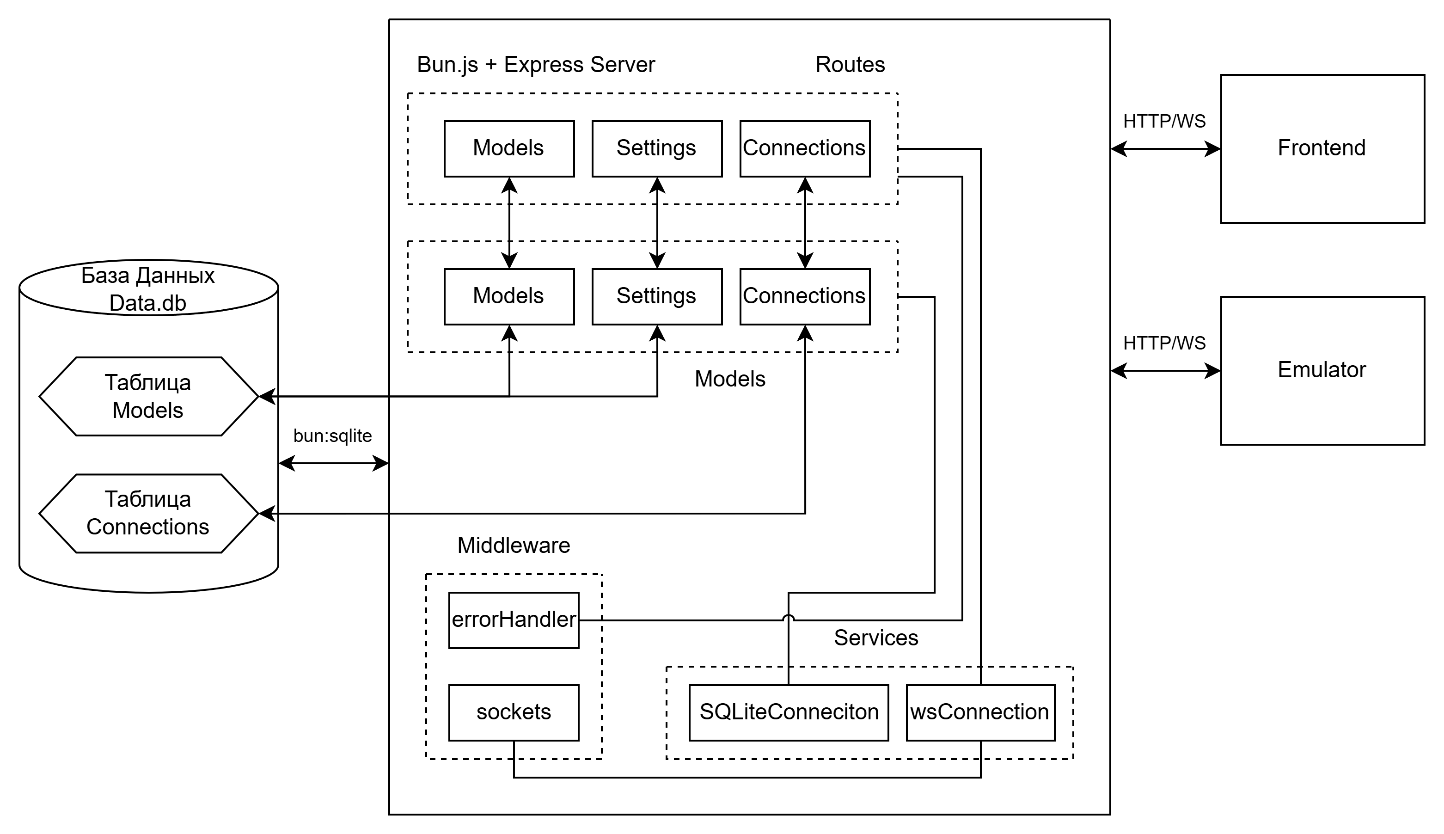
Бэкенд реализован на основе Bun.js, обеспечивающего высокую производительность за счёт нативной поддержки SQLite. Основным фреймворком выбран Express из-за простоты его интеграции с JavaScript-экосистемой и широкой поддержки паттерна middleware. Для взаимодействия в реальном времени между фронтендом и эмулятором используется Socket.IO, позволяющее передавать команды и состояния с минимальной задержкой. Архитектура сервера организована по модульному принципу, разделяя функциональность на маршруты (routes), модели (models), сервисы (services) и middleware (middleware). Основной входной точкой является файл src/index.ts, который инициализирует сервер и настраивает middleware, включая CORS, body-parser и cookie-parser.

Сервер выполняет следующие функции:

* Обработка HTTP-запросов для управления моделями и подключениями через REST API;
* Хранение данных моделей и подключений в базе данных SQLite (data.db);
* Пересылка команд и состояний между фронтендом и эмулятором через Socket.IO;
* Проверка доступности эмулятора через HTTP-запросы;

Модульная структура упрощает поддержку и масштабирование системы, позволяя независимо обновлять отдельные компоненты, такие как маршруты или обработчики Socket.IO.

На рисунке 5.1 представлена архитектура сервеной части. Отдельно выделены блоки маршрутов (routes), моделей (models), сервисов (services) и промежуточное программное обеспечение (middleware). Показано их взаимодействие междну собой.

****Рисунок 5.1 – Архитектура серверной части

Маршруты – предостовляют функциональность путей для HTTP запросов

Модели – представляют набор функций для работы с базой данны

Сервисы – обеспечивают соединение с базой данных, и веб-сокет соединение между сервером, клиентами и эмуляторами.

Промежуточное программное обеспечение – предоставляет функции для обработки данных или ошибок, внутри маршрутов, моделей и сервисов.

* 1. Разработка REST API

Для управления данными 3D-моделей и подключениями к эмулятору разработан REST API, реализованный с использованием Express. API включает три основных маршрута, определённых в файлах src/routes/models.ts, src/routes/connections.ts и src/routes/settings.ts. Основные эндпоинты обеспечивают CRUD-операции (создание, чтение, обновление, удаление) для моделей и подключений, а также проверку доступности эмулятора.

Основные маршруты:

**/api/models**:

* GET /: получение списка имён моделей с их идентификаторами;
* GET /:modelName: получение бинарных данных модели в формате application/octet-stream;
* POST /: загрузка новой модели с использованием multer для обработки бинарных файлов;
* PUT /:modelName: обновление модели (данных или настроек);
* DELETE /:modelName: удаление модели.

**/api/connect**:

* GET /?page=X&perPage=Y: постраничное получение списка подключений;
* POST /: добавление нового подключения (имя, IP, порт);
* PATCH /:id: обновление подключения по идентификатору;
* DELETE /:id: удаление подключения;
* POST /ping: проверка доступности эмулятора по IP и порту.

**/api/json**

* GET /:modelName: получение JSON-настроек модели для управления взаимодействием в веб-интерфейсе.

Для обработки файлов в POST /api/models используется библиотека multer, которая позволяет загружать бинарные данные модели с ограничением размера 50 МБ. Ошибки обрабатываются централизованным промежуточным программным обеспечением errorHandler возвращающим стандартизированный JSON-ответ с кодом 400 и описанием ошибки. Безопасность запросов обеспечена использованием CORS с белым листом (whitelist) для доменов используемых клиентской частью и эмуляторами, а также проверкой входных данных в маршрутах.

Пример запроса для загрузки модели представлен в листинге 5.2

Листинг 5.1 – Пример запроса для загрузки

|  |
| --- |
| POST /api/models  Content-Type: multipart/form-data  {  "name": "xray\_model",  "settings": "{\"controls\": {\"joystick\": {\"x\": 0, \"y\": 0}}}",  "data": [binary glb file]  } |

Древовидная диаграмма, показывающая иерархию маршрутов API представлена на рисунке 5.2

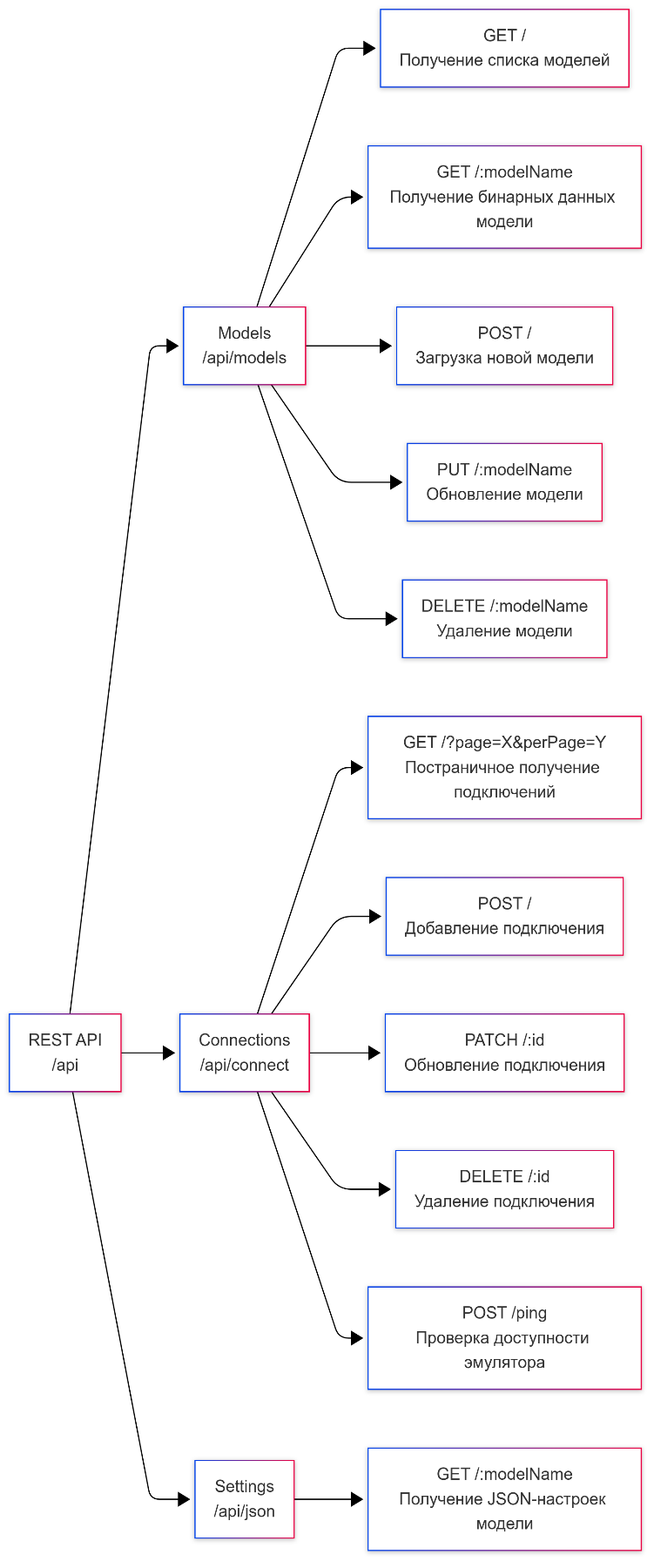


Рисунок 5.2 – Древовидная диаграмма, показывающая иерархию маршрутов API

* 1. Реализация базы данных

Для хранения данных 3D-моделей и подключений используется реляционная база данных SQLite, размещённая в файле data.db. Инициализация базы выполняется в файле db/init-db.ts, который создаёт две таблицы:

**Таблица models**:

* id (INTEGER, автоинкремент, первичный ключ): уникальный идентификатор модели;
* name (TEXT, уникальное): имя модели;
* settings (TEXT): JSON-строка с настройками взаимодействия (например, диапазоны для джойстика);
* data (BLOB): бинарные данные модели в формате glTF;

**Таблица connections**:

* id (INTEGER, автоинкремент, первичный ключ): идентификатор подключения;
* name (TEXT): название подключения;
* ip (TEXT): IP-адрес эмулятора;
* port (TEXT): порт эмулятора;
* Уникальный индекс на поля ip и port для предотвращения дублирования.

Доступ к базе осуществляется через модуль src/services/SQLiteConnection.ts, который использует библиотеку bun:sqlite для выполнения параметризованных запросов. Это обеспечивает защиту от SQL-инъекций, как, например, в функциях addConnection и updateModelByName. Модули в src/models (например, connections.model.ts, models.model.ts) предоставляют функции для CRUD-операций, такие как getConnectionsPaginated для постраничного вывода подключений или getModelByName для получения бинарных данных модели.

Пример параметризованного запроса для добавления подключения представлен в листинге 5.3

Листинг 5.3 – Пример запроса для загрузки

|  |
| --- |
| const query = db.query(`INSERT INTO connections (name, ip, port) VALUES (?, ?, ?)`)*;*  query.run(name, ip, port)*;* |

База данных оптимизирована для небольшого объёма данных, характерного для демонстрационного приложения, обеспечивая время выполнения запросов менее 10 мс для таблиц с до 1000 записей.

ER-диаграмма, показывающая таблицы models и connections представлена на рисунке 5.3

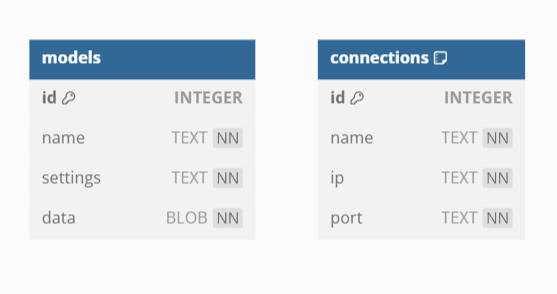


Рисунок 5.3 – ER-диаграмма, показывающая таблицы models и connections

* 1. Интеграция с Socket.IO

Для обеспечения взаимодействия в реальном времени между фронтендом и эмулятором сервер использует библиотеку Socket.IO, реализованную в файле src/middleware/sockets.ts. Socket.IO настроен с поддержкой CORS для тех же доменов (localhost:8045, 8046) и обрабатывает подключения через функцию socketManager. Основные события, обрабатываемые сервером:

* **connectHardware**: устанавливает соединение с эмулятором по указанным IP и порту, используя WsConnect из src/services/wsConnection.ts. Применяется async-mutex для предотвращения конфликтов при множественных подключениях;
* **clientCommand**: пересылает команды (например, { command: "set", path: "table\_x", value: 10.5 }) от фронтенда к эмулятору;
* **clientState**: передаёт состояния оборудования (например, isControlsEnabled) от фронтенда к эмулятору;
* **getCurrentParameters**, **getModel**: запрашивают текущие параметры или имя модели у эмулятора.

**События от эмулятора**: command, state, setParameters пересылаются фронтенду как clientCommand, clientState, clientSetParameters.

Механизм async-mutex гарантирует, что только одно соединение с эмулятором активно в любой момент времени, предотвращая дублирование команд. Задержка передачи данных через Socket.IO составляет менее 100 мс, что обеспечивает отзывчивость интерфейса при управлении 3D-моделью.

Socket.IO также поддерживает автоматическое переподключение, что повышает надёжность системы при временных сбоях сети.

На рисунке 5.4 представлена диаграмма последовательности (Sequence Diagram), показывающая взаимодействие фронтенда, бэкенда и эмулятора с помощью веб-сокетов. Этапы: 1) Фронтенд отправляет clientCommand на бэкенд; 2) Бэкенд пересылает command эмулятору; 3) Эмулятор возвращает setParameters; 4) Бэкенд отправляет clientSetParameters фронтенду.

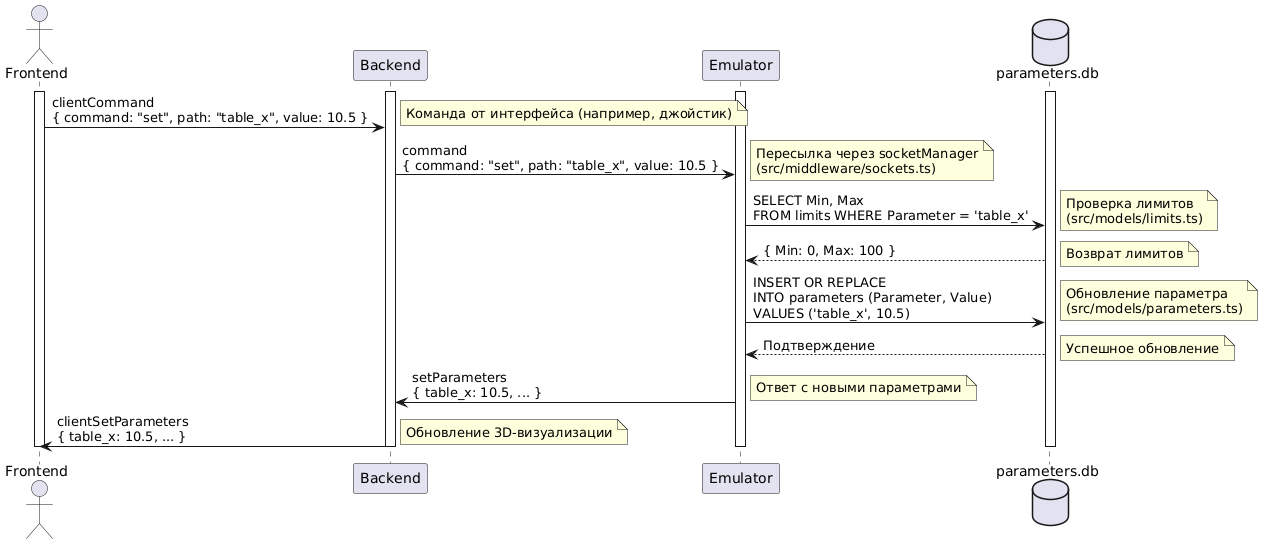


Рисунок 5.4 - диаграмма последовательности

* 1. Выводы

Разработанный сервер обеспечивает надёжное управление 3D-моделями и подключениями, а также реальное время взаимодействия с эмулятором рентгеновского аппарата. REST API, реализованное на Express, поддерживает CRUD-операции с высокой производительностью, а Socket.IO гарантирует отзывчивость системы с задержкой менее 100 мс. SQLite база данных эффективно хранит данные, а параметризованные запросы обеспечивают безопасность.

1. Создание сервера эмулятора рентгеновского аппарата.

Эмулятор рентгеновского аппарата является ключевым компонентом системы, обеспечивающим реалистичную имитацию работы телеуправляемого медицинского оборудования. Он обрабатывает команды и состояния, поступающие от серверной части (бэкенда), обновляет параметры в базе данных с учётом физических ограничений и возвращает результаты для синхронизации с 3D-визуализацией в веб-интерфейсе. Эмулятор реализован на Bun.js, используя Express для минимального HTTP API и Socket.IO для взаимодействия в реальном времени. Для хранения текущих параметров и их лимитов применяется база данных SQLite (parameters.db). В данной главе описаны архитектура эмулятора, настройка базы данных, реализация обработки команд, интеграция с системой и результаты тестирования.

**6.1 Архитектура эмулятора**

Эмулятор реализован как отдельный сервер, работающий на Bun.js, который обеспечивает высокую производительность нативной поддержке SQLite. Для обработки HTTP-запросов используется фреймворк Express, а для реального времени — библиотека Socket.IO. Архитектура эмулятора организована модульно, с разделением функциональности на следующие компоненты:

* Маршруты (src/index.ts): минимальное HTTP API для проверки доступности (GET /ping).
* Обработчики Socket.IO (src/middleware/sockets.ts): управление командами, состояниями и параметрами.
* Модели базы данных (src/models/limits.ts, src/models/parameters.ts): функции для работы с таблицами limits и parameters.
* Сервисы (src/services/CommandProcess.ts, src/services/SQLiteConnection.ts): логика обработки команд и подключение к базе данных.

Основной файл src/index.ts инициализирует сервер на порту 12537 (или из переменной окружения PORT), настраивает middleware (CORS, body-parser, cookie-parser) и запускает Socket.IO с обработкой соединений через socketManager. Эмулятор выступает как имитатор оборудования, принимая команды (например, установка позиции стола) от бэкенда, проверяя их на соответствие лимитам и возвращая обновлённые параметры для отображения в веб-интерфейсе.

Эмулятор поддерживает следующие функции:

* Обработка команд set (установка значения) и add (инкремент параметра).
* Управление состояниями оборудования (isEnabled, isControlsEnabled, isEmergencyStoped).
* Хранение параметров и лимитов в базе данных SQLite.
* Передача данных в реальном времени через Socket.IO.

Модульная структура упрощает добавление новых функций, таких как симуляция ошибок оборудования или поддержка сложных сценариев движения.

На рисунке представлена архитектура эмулятора. Отдельно выделены блоки маршрутов моделей (models), сервисов (services) и промежуточное программное обеспечение (middleware), а также блок Command Processor, ответсвенный за исполнение и обработку команд. Показано их взаимодействие междну собой.

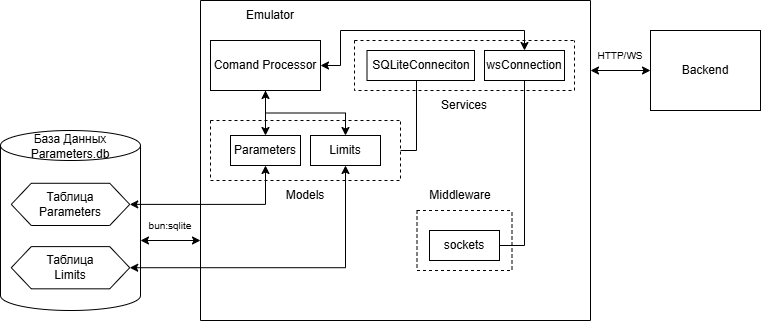


Рисунок 6.1 – Архитектура эмулятора

1. Проверка работоспособности приложения
   1. Цели и методы тестирования

Проверка работоспособности приложения проводилась для подтверждения корректной реализации всех компонентов: отображения 3D-модели, пользовательского интерфейса, серверной части и базы данных. Основные цели тестирования:

* Проверка загрузки и отображения 3D-модели без ошибок.
* Проверка функциональности элементов управления (джойстика, кнопок, полей ввода).
* Проверка взаимодействия фронтенда с бэкендом через API.
* Проверка производительности и стабильности приложения.

Тестирование проводилось в среде разработки на локальном сервере (http://localhost:8045 для фронтенда, http://localhost:8046 для бэкенда) с использованием браузера Google Chrome версии 126.

* 1. Тестирование отображения модели

Для проверки отображения модели использовалась страница Viewer.tsx, где:

* Выбиралась модель из выпадающего списка (SLSelect).
* Проверялось корректное отображение модели в сцене с учётом освещения и теней.
* Проверялось управление моделью через TrackballControls (вращение, масштабирование, перемещение).

Результаты:

* Модель в формате glb успешно загружалась через API (/api/models/:modelName).
* Освещение и тени отображались корректно, обеспечивая реалистичную визуализацию.
* Управление моделью работало плавно, без задержек.

На рисунке 4 представлено отображение модели.

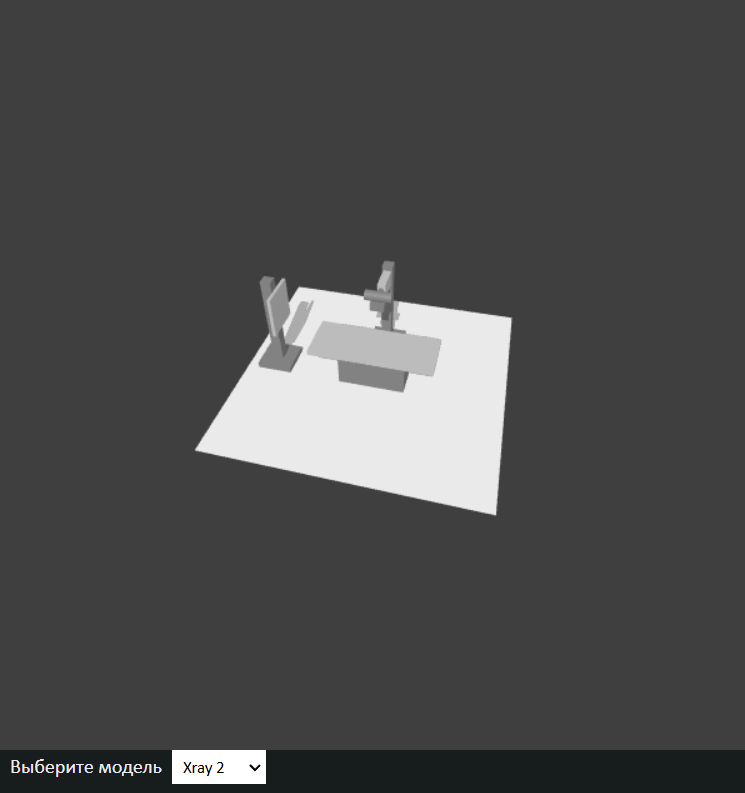


Рисунок 4 – Отображение модели

* 1. Тестирование пользовательского интерфейса

Интерфейс тестировался на странице Viewer.tsx с включённым режимом управления (modelControlsEnable = true):

* **Джойстик**: проверялось изменение положения и поворота компонентов модели при движении джойстика. Параметры изменялись с шагом 0.1 в пределах заданных ограничений.
* **Кнопка установки**: проверялось срабатывание кнопки (SSetButton) для установки предопределённых значений. Переход к целевым значениям происходил плавно.
* **Поля ввода**: проверялось изменение значений положения и поворота через числовые поля. Ввод значений за пределами ограничений блокировался.

Результаты:

* Все элементы управления работали корректно, изменения отражались на модели в реальном времени.
* JSON-конфигурация загружалась без ошибок и корректно применялась к интерфейсу.

Обнаруженные проблемы:

* Незначительные задержки при частом использовании джойстика из-за интервального обновления (100 мс). Оптимизация не проводилась, так как задержки не влияли на общее восприятие.

На рисунке 5 представлено отображение интерфейса.

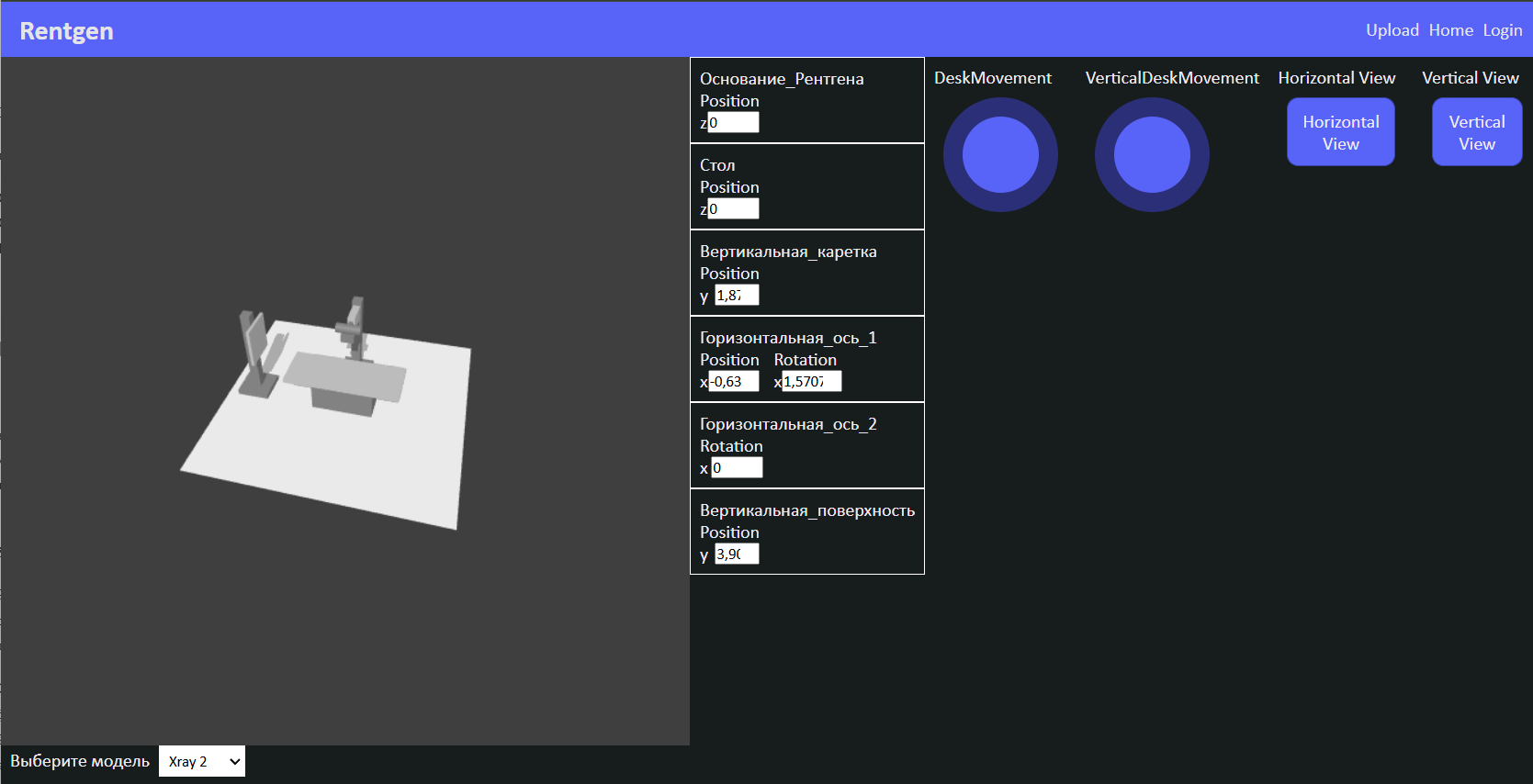


Рисунок 5 – отображение интерфейса.

* 1. Тестирование серверной части

Серверная часть тестировалась через Postman и прямые запросы из фронтенда:

* **GET /api/models**: возвращал список всех моделей с их именами и ID.
* **GET /api/models/:modelName**: возвращал бинарные данные модели.
* **POST /api/models**: проверялась загрузка новой модели через форму AddModel.tsx.
* **GET /api/json/:modelName**: возвращал JSON-конфигурацию для модели.

Результаты:

* Все endpoints работали корректно, возвращая ожидаемые данные.
* Загрузка новых моделей через multer и их сохранение в SQLite выполнялись без ошибок.
  1. Тестирование базы данных

База данных SQLite (db/models.db) тестировалась на:

* Сохранение моделей (INSERT INTO models).
* Получение моделей по имени (SELECT \* FROM models WHERE name = ?).
* Получение списка моделей (SELECT id, name FROM models).

Результаты:

* Все запросы выполнялись корректно, данные сохранялись и извлекались без ошибок.
* Производительность базы данных была достаточной для небольшого объёма данных.
  1. Проверка производительности

Производительность тестировалась с использованием инструментов разработчика Chrome:

* Средний FPS при взаимодействии с моделью составлял 50–60 кадров в секунду.
* Время загрузки модели (около 2 МБ) составляло менее 1 секунды при локальном подключении.
* Память приложения оставалась стабильной благодаря очистке ресурсов при размонтировании компонентов.
  1. Выводы

Проверка работоспособности подтвердила, что приложение успешно выполняет поставленные задачи: отображает 3D-модель, предоставляет интерактивный интерфейс и обеспечивает стабильное взаимодействие с сервером и базой данных. Обнаруженные проблемы были незначительными и не влияли на основной функционал. Приложение готово к использованию в образовательных и демонстрационных целях.

# ЗАКЛЮЧЕНИЕ

Разработанное веб-приложение успешно решает задачу отображения взаимодействия настроек с 3D-визуализацией телеуправляемого рентгеновского аппарата. В ходе выполнения работы были достигнуты следующие результаты:

* Создана реалистичная 3D-модель рентгеновского аппарата в Blender с экспортом в формат glTF 2.0.
* Реализован веб-интерфейс на основе React и Three.js, обеспечивающий интерактивное управление моделью через джойстик, кнопки и числовые поля.
* Разработан сервер-эмулятор на Node.js с Express, поддерживающий API для работы с моделями и их конфигурациями.
* Настроена база данных SQLite для хранения данных моделей, обеспечивающая простоту и производительность.
* Проведено тестирование, подтвердившее работоспособность всех компонентов приложения.

Использование современных технологий, таких как Three.js, WebGL и Redux Toolkit, позволило создать масштабируемое и производительное решение. Применение JSON для конфигурации управления обеспечило гибкость и простоту настройки, а выбор SQLite минимизировал требования к серверным ресурсам.

Приложение имеет практическую ценность для образовательных целей, обучения медицинского персонала и демонстрации возможностей телеуправляемых рентгеновских систем. В перспективе возможно расширение функциональности, например:

* Добавление поддержки анимаций для модели.
* Внедрение проверки типов загружаемых файлов на сервере.
* Оптимизация интерфейса для мобильных устройств.
* Интеграция дополнительных симуляций работы рентгеновского аппарата.

Данная работа способствует развитию технологий 3D-визуализации в веб-приложениях и подтверждает перспективность их применения в медицинской и образовательной сферах.